PARALLEL ARCHITECTURES TO IMPROVE A GA BASED REAL-TIME SYSTEM FOR TRADING THE STOCK MARKET.

IVÁN CONTRERAS FERNÁNDEZ - DÁVILA

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Máster en Ingeniería de la Industria

June, 20th 2011
Score:9,5

Directors:

Jose Ignacio Hidalgo Perez (UCM)
Laura Núñez-Letamendia (IE Business School)

Autorización de difusión

Iván Contreras Fernández - Dávila

June, 20th 2011

El abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "PARALLEL ARCHITECTURES TO IMPROVE A GA BASED REAL-TIME SYSTEM FOR TRADING THE STOCK MARKET", realizado durante el curso académico 2010-2011 bajo la dirección de Jose Ignacio Hidalgo Perez y con la colaboración externa de dirección de Laura Nuñez Letamendi del I.E, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen en castellano

La investigación y el desarrollo de sistemas de trading son cada vez más frecuentes ya que pueden alcanzar un alto potencial en la predicción de los movimientos bursatiles. El uso de estos sistemas permite manejar una enorme cantidad de datos relacionados con factores que afectan directamente al rendimiento de las inversiones (variables macroeconómicas, información de las compañías, indicadores industriales, variables de mercado, etc.), además evita las reacciones psicológicas asociadas a la inversión en los mercados financieros. Los movimientos de los mercados bursátiles son continuos a lo largo de cada día, lo que reclama que los sistemas de trading deban ser apoyados por motores de analisis muy potentes, ya que la cantidad de datos necesarios para hacer frente a unas buenas predicciones crece, mientras que el tiempo de respuesta se acorta. Numerosos estudios documentan el uso de algoritmos genéticos (AG) como eje principal de los sistemas de trading. Los resultados experimentales proporcionados en este documento muestran diferentes formás de combinar el uso de AG y sistemas de paralelización. La paralelización mediantes las técnicas propuestas proporcionan una cuantiosa aceleración en la potencia y la capacidad de búsqueda de los AG para este tipo de aplicaciones financieras. Por otra parte, el analisis previo a la paralelización nos permite implementar mejoras para las anteriores aproximaciones de AG. Respecto a los resultados de inversión, se pueden demostrar un 870% de ganancias para el S&P 500 en un plazo de 10 años (1996-2006), cuando la ganancia media del índice S&P 500 en el mismo período fue de 273%.

Palabras clave

Boinc, computación distribuida, sistema de trading, algoritmo genético, Jacket, GPU

Abstract

Research and development of trading systems are becoming more frequent as they can reach a high potential in the prediction of market movements. The use of these systems allows to manage a huge amount of data related to the factors affecting investments performance (macroeconomic variables, company information, industrial indicators, market variables, etc.) while avoids the psychological reactions of traders when they invest in financial markets. The movements in the stock markets are continuous throughout each day, which requires that trading systems should be supported by very powerful engines, since the amount of data to deal with grows while the respond time required to support trades gets shorter. Numerous studies document the use of genetic algorithms (GA) as the engine driving mechanical trading systems. The experimental results provided in this paper show different ways of combining the use of AG and parallelization systems. Parallelization using the proposed techniques provide a substantial acceleration in the power and capacity of the GA search for this type of financial applications. Moreover, a previous analysis of the paralellization allows us to implement improvements to the previous approaches AG. With regard to investment results, we demonstrate a 870% of earnings for the S&P 500 over a period of 10 years (1996-2006), when the average gain in the S&P 500 over the same period was 273%.

Keywords

Boinc, grid computing,trading system, genetic algorithm, Jacket, GPU

Contents

List of Figures

List of Tables

Agradecimientos

Principalmente, quiero agradecer la preocupación y paciencia que han tenido conmigo durante la realización de este trabajo, mis tutores: Jose Ignacio Y Laura. Gracias por vuestra ayuda y por compartir vuestros conocimientos.

Dedicatoria

Este trabajo está dedicado completamente a mis padres. Gracias por estar siempre ahi.

Chapter 1

Introduction

The growing availability of data for financial markets and companies, together with the increasing complexity of socioeconomic and financial environment, tragically illustrated by the financial crisis, makes it more difficult the decision making process for real time investments in stock markets. Complexity in this sense derives from the difficulty of the assets valuation process due that each type of asset requires addressing its valuation with a separate approach to be able to modeling the factors affecting its performance in order to anticipate its risk adjusted return. The huge number of potential interrelated factors and their changing time patterns, affecting financial assets, make the investment process remains a challenge. In the past years interest in mechanical or automatic trading systems has spread due to several factors:

- (i) the explosive amount of information available for companies and markets that makes it hard the analysis of more than a few stocks or assets
- (ii) the advance of inexpensive computing power
- (iii) the tremendous expansion of the globalization process symbolized for instance by the tremendous development of the "hedge funds industry", a class of funds which invest in any kind of assets around the World (stocks, indexes, bonds, commodities, currencies, etc.)

- (iv) the attempt to avoid the psychological aspects that biases the investment process (known in the literature as behavioral finance (32)

Mechanical trading systems are systems based on rules that use market, business or macroeconomic information embedded in algorithms that look for the best combination of these rules to drive the stock trades in an attempt of obtaining the maximum possible return for a period. Mechanical trading systems have evolved from very simple *If-then* algorithms to more sophisticated models that use methods like artificial intelligence, chaos theory, fractals, evolutionary algorithms, nonlinear stochastic representations, econo-physics models, etc., which are ultimately based on market, fundamental or macroeconomic data.

In particular previous studies document the use of Genetic Algorithms to design and optimize automatic trading systems for the Stock Market (see 2; 27; 26; 19; 28). Furthermore, the investment industry has now begun to use GAs and other kinds of evolutionary algorithms to build automatic trading systems. Not only the biggest investment firms have these kind of systems, but smaller firms are also beginning to access GA tools through marketed software packages. One example of these software packages is "*GeneHunter*", marketed by *MBAWare* an American Company (http://www.mbaware.com) located in Arlington, Virginia in the Washington DC metro area, with more than 2500 customers in 40 countries. The firm states on its website that "*GeneHunter* is a powerful software solution for optimization problems which utilizes a state-of-the-art genetic algorithm methodology "pointing out that "GeneHunter has been used to generate rules to predict a rise in the NYSE index". This is just one example of an increasing number of GA-based software for investment management. In (28) the authors describe a trading system designed with GAs that use different kind of rules with market, macroeconomic and companies information, which is applied to trade, in a daily base, the companies belonging to the S&P 500 index. One of the difficulties the authors claim is the computational time required for training the trading system with daily data of stocks prices. This restriction is even more critical when we take into account that the majority of traders invest in an intra-day base (what means that the investment

positions are canceled during the same day). Therefore, the operative in financial markets seems to recommend the use of an even shorter frequency in data when training and applying mechanical trading systems.

However, the focus on intra day instead daily data cause some difficulties for the design and application of these trading systems because of the shorter time of respond we require, since it is not possible to wait for hours to obtain the investment decision result provided by the mechanical trading system when the investments have to be done continuously during the day. The necessity of speed up the GA process, to get in time good results for this kind of trading applications, as well as its possibilities of parallelization pushed us to think about new ways of implementing these applications.

In this work we propose to combine the use of two differents parallel computer architectures to speed up the functioning of a GA used to design trading systems to invest in stocks. First, we have used a corporative grid, and later, a graphics device, both plataforms have been used for to obtain a profit in computation time. We can found lot of cases in the literature where they used this plataforms for to optimize time. Experimental Results show how the combination of the GA and the parallel architectures allows us to obtain solutions for real time (or intra-day data) investment decision. It is very difficult to have access to intra-day data since the commercial databases for stock prices and stock exchange markets do not provide frequency data inferior to that of daily prices. The way to accede to intra-day data is through an investment bank, and usually they do not like to publish results obtained in joined research projects. Nevertheless since the only different between a trading-system based on intra-day data versus daily data is the respond time required, that when dealing with intra-day data should be rather shorter, we apply our trading systems to daily data, because of the mentioned difficulty of accede to intra-day data. We show how we can use and configure the GA on the parallel architectures in order to analyze results for different companies at once.

The rest of the document is organized as follows.

In the next chapter, Chapter 2, we formulate the investment problem describing the proposed trading system to cope with it. Furthermore, the Chapter 4 introduces with some detail the implementation of the original GA for the developed application of a mechanical trading system.

In Chapter 3 we explain the parallelization in the Boinc platform. We take a look to the general achitecture of Boinc and later we carry out several test of performance.

Chapter 4 describes the general parallel architecture, in this case the Cuda architecture. Moreover we show the parallelization process with high detail. In the last sections of this chapter we execute the parallized program in the computer graphic card and we show the experimental results.

Finally, we summarize and conclude in Chapter 5, where we show both parallel architectures in opposition. In addition, in the last pages we can find several appendixes where we explain details for some parts of the work for develop this document.

Chapter 2

Trading System

2.1 Formulation of the investment problem

The investment problem is defined as to maximize return (or risk adjusted return) for a specific time period when investing long or short-sell in a financial asset (in our case a stock). Since the performance of investment decisions in stock markets is influenced by a wideness of factors of different type: political, macroeconomic, regulatory, local, international, etc. that are uncertain, there is not a single and perfect rule with a specific parameter or threshold value that can be used to maximize future returns, but a broad set of potential rules which combine indicators, representing different factors and driven by a range of values in their parameters. Therefore, the investment problem consists first of finding the best combination of indicators and second of fine-tuning the parameters for these indicators to obtain the maximum return when applied to the investment decision making in a stock. Thus, the input set of variables for a trading system consists of the indicators to be used as investment criteria and the parameters being the threshold values for these indicators, while the output is the return obtained by the trading system this way defined.

2.2 Sample data

We use diferents data for the trading system, the initial sample of firms comprises all companies included in the S&P 500 Index Constituent List for at least two quarters during the

period January 1986 to December 2006, with the exception of those belonging to the finance, insurance, and real estate industries. This gives us a total of 834 companies.

The program need between 5 and 10 years of enterprise information to fine-tune the fundamental trading system. Firms with less than consecutively 20 quarters non-missing value before the listing year are deleted from the list. The final sample size is 599 companies and the sample period ranges from 1976 Q1-2006 Q4, among which only 24 companies survive throughout the full period.

Finally the data provide 332.710 observations for quarterly fundamental data and 7.157.320 observations for daily technical data. The source of the data is from Compustat and CRSP databases.

2.3 Genetic algorithm

A genetic algorithm (GA) is a search heuristic that mimics the process of natural evolution. As defined by Koza (see 33), "The genetic algorithm simulates Darwinian evolutionary processes and naturally occurring genetic operations on chromosomes"..."The genetic algorithm is a highly parallel mathematical algorithm that transforms a set (population) of individual mathematical objects, each with an associated fitness value, into a new population using operators patterned after the Darwinian principle of reproduction and survival of the fittest and after naturally ocurring genetic operations...". This heuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms, which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

Figure 2.1 represents the mainly work flow in the genetics algorithms. The functioning of the GA methodology is driven by different parameters: the crossover and mutation percents, the scaling of fitness function in the reproduction process, the initial population and

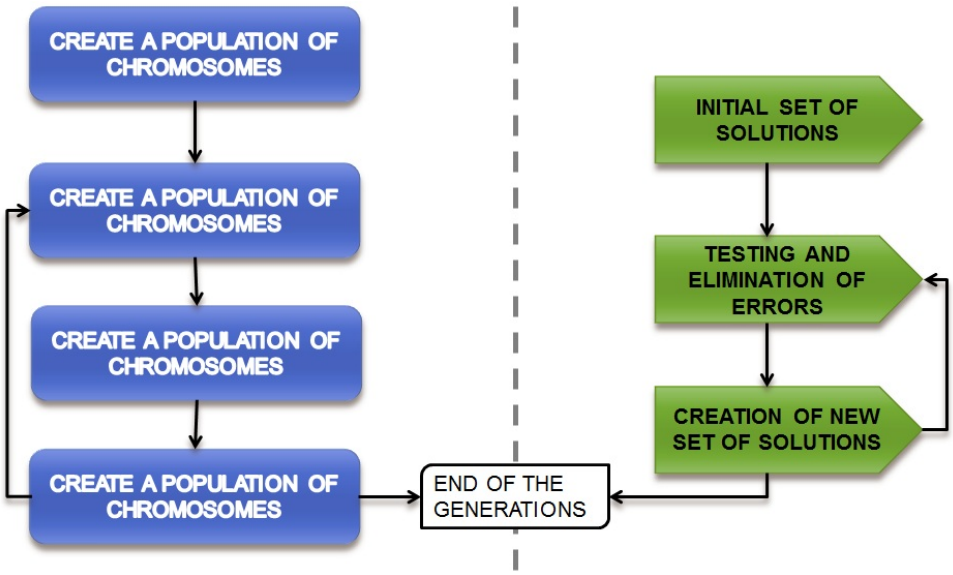


Figure 2.1: *Typical genetic algorithm work flow*

the numbers of generations. We considered important to mention that these parameters depend on a particular problem, this mean that the good results for a combination of a specific values cannot be extrapolated to others genetic algorithm problems.

As it is well known, the design of a GA involves some key factors that will depend heavily on the characteristics of the problem at hand. One of them is the chromosome encoding used for representing the solutions by means of some type of code (binary, real, etc.). For codifying the chromosomes we adopted the following approach: Our GA is used to select the threshold values of the 4 indicators that comprise the trading systems or investment rules (see 2.2, 2.3), and we use binary code chains like these: [0101-0011, 0010-1111, 0110, 1001-0001] where the first 8 genes correspond to the threshold value of the short selling and long positions for the company multiple price ratio preselected indicator, the next 8 match to those of the company improvement or efficiency preselected ratio, the next 4 to those of the leverage indicator (this type of indicator is only employed to generate short selling signals, that is why the chain for the parameters only contains 4 genes instead of 8)

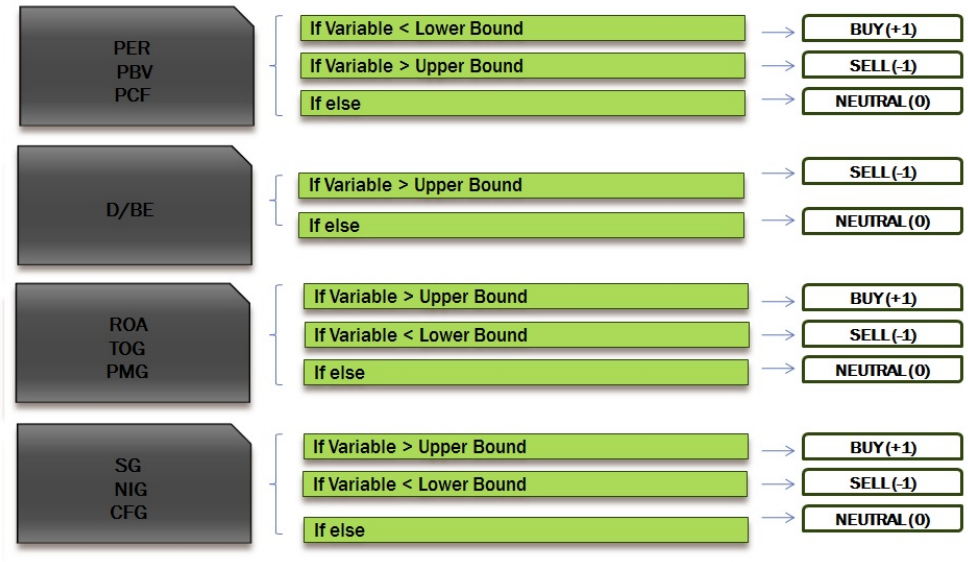


Figure 2.2: *Fundamental Trading Rules*

and the last 8 correspond to those of the company growth selected indicator. Hence, our chromosomes of trading systems comprise 28 binary genes giving a total search space of $2^{28} = 268435456$ possible combinations, implying that the analysis of more than 2.5 millions of potential combinations of indicators with different threshold values would reach only 1% of the full search space.

For selection we use the roulette wheel (16) method initially (on Boinc system) and tournament selection (on the GPU). Crossover and mutation are implemented as usual based on one-point crossover and mutation. The GA choice of threshold values is driven by the value of the fitness function (the accumulated return obtained computed as described in section 2.4.1) when applying the trading systems to the sample data.

Although this method has given promising results, the main problem of the initial approximation is the execution time. In this sense with a sequential execution of the GA we are not able to apply our proposal to real time problems with intra-day data. In the following sections we explain how we can approach this problem by using several parallel implementations. For this purpose we made first an analysis of the execution time for the whole

Figure 2.3: *Technical Trading Rules*

program and, based on the results, we implemented several important changes not only in the structure of the program, but also in the genetic operators.

2.4 Two types of financial analysis

When the objective of the analysis is to determine what stock to buy and at what price, there are two basic methodologies: fundamental analysis and technical analysis. Investors can use any or all of these different but somewhat complementary methods for stock picking. For example many fundamental investors use technicals for deciding entry and exit points. Many technical investors use the analysis fundamental to limit their universe of possible stock to good companies.

Figure 2.4 shows the typical difference in a middle term investment. The Y axis represent the price of a company in a stock market, and the X axis represents the price evolution in a year. The red arrows represents the possible signals of sell and buy that we could have done in operations using the technical analysis. On other hand, if we had used the fundamental

Figure 2.4: *Typical differences between fundamental and technical analysis.*

analysis, we would have done the green marks as signals of sell and buy. When the arrows point down means "sell" and when these arrows are pointing upward means "buy". As we can observe in the Figure 2.4, the fundamental analysis obtains less benefit, to change for a less intervention in the process of investments, that mean, more freedom for the investors.

2.4.1 Fundamental analysis

Fundamental analysis maintains that markets may misprice a security in the short run but that the "correct" price will eventually be reached. Profits can be made by trading the mispriced security and then waiting for the market to recognize its "mistake" and reprice the security. Fundamental analysis of a business involves analyzing its financial statements and health, its management and competitive advantages, and its competitors and markets. When applied to futures and forex, it focuses on the overall state of the economy, interest rates, production, earnings, and management.

Millions of indicators can be used, thus the first step to solve the investment problem is to select the indicators to be included in the decision making trading system. In this document

we follow Jiang et al. (2009) who carried out an exhaustive analysis of the investment literature to pre-select the set of indicators to be used in their trading systems driven by a GA: price-earnings ratio, price-cash flow ratio, price-book value ratio; price-earnings ratio (PER), price-book value ratio (PBV), price-cash flow ratio (PCF), debt over book equity (D/BE), sales growth (SG), net income growth (NIG), cash flow growth (CFG), return on assets (ROA), turnover growth (TOG), and profit margin growth (PMG). All these indicators have been reported to be useful in the literature on investments (see 3; 5; 9; 11; 15; 14; 31, among others). The formulation for their initial set of indicator is presented in Table 2.1. For a broader description see the original paper by these authors (19).

To compute these indicators (19) obtained from the COMPUSTAT database, quarterly data on nine items from the companies financial statements (COMPUSTAT codes in parenthesis) for the period January 1986 to December 2006:

- total assets (ATQ)
- total common ordinary equity (CEQQ)
- cash and short-term investments (CHEQ)
- debt in current liabilities (DLCQ)
- total long-term debt (DLTTQ)
- total depreciation and amortization (DPQ)
- net income / loss (NIQ)
- total revenue (REVTQ)
- common shares outstanding (CSHOQ).

Prices (PRCCQ) were obtained from the Center for Research in Security Prices (CRSP) on a quarterly basis and were adjusted by stock splits and stocks dividends. To ensure

Table 2.1: *Formulation of indicators used for fundamental analysis*

Indicators representing Company Price Multiples		
Name	Description	Formula
PER_t	Price Earning Ratio	$\frac{PRCCQ_t \cdot CSHOQ_t}{NIQ_{t-1}}$
PBV_t	Price Book Value	$\frac{PRCCQ_t \cdot CSHOQ_t}{CEQQ_{t-1}}$
PCF_t	Price Cash Flow	$\frac{PRCCQ_t \cdot CSHOQ_t}{NIQ_{t-1}+DPQ_{t-1}}$
Indicators representing Company Leverage		
Name	Description	Formula
D_{t-1}/BE_{t-1}	Debt / Book Value of Common Equity	$\frac{DLTTQ_{t-1}+DLCQ_{t-1}-CHEQ_{t-1}}{CEQQ_{t-1}}$
Indicators representing Company Growth		
Name	Description	Formula
SG_{t-1}	Growth in Sales	$\frac{REVTQ_{t-1}}{REVTQ_{t-2}} - 1$
NIG_{t-1}	Growth in Net Income	$\frac{NIQ_{t-1}}{NIQ_{t-2}} - 1$
CFG_{t-1}	Growth in Cash Flow	$\frac{NIQt-1+DPQ_{t-1}}{NIQ_{t-2}+DPQ_{t-2}} - 1$
Indicators representing Company Improvement or Efficiency		
Name	Description	Formula
PMG_{t-1}	Growth in Profit Margin	$\frac{NIQ_{t-1}/REVTQ_{t-1}}{NIQ_{t-2}/REVTQ_{t-2}} - 1$
TOG_{t-1}	Growth in Turnover Ratio	$\frac{REVTQ_{t-1}/ATQ_{t-1}}{REVTQ_{t-0}/ATQ_{t-0}} - 1$
ROA_{t-1}	Return on Asset	$\frac{NIQ_{t-1}}{ATQ_{t-1}}$

that the items used to compute the indicators at quarter t were known to the market as of quarter t, they used the data on quarter t-1 for all of them, except for the common shares outstanding and prices. Therefore financial statement data for period t-1 was matched with prices and common shares outstanding data at time t.

Since the above 10 indicators can be clustered around 4 classes that represent similar features of the Company performance, Jiang used a GA to pre-select one indicator from each cluster to signal the long and short selling investment decisions, allowing to choose different indicators for long trades than for short selling trades. The final selected indicators for short positions were: PCF, D/BE, SG and TOG while the indicators used for long positions were PBV, SG and ROA (they did not use the leverage indicator for investing long).

Parameters

The above mentioned indicators are applied to the investment process related to a threshold or parameter value. For instance, the PER can be used as follows: take a long position in the company stock if the PER is below 10 / invest short in the company stock if the PER is above 20. Therefore, it is necessary to fine-tune the parameter for each one of the indicators and for both positions long and short-sell, within a range that can be defined following the investors practices. The trading system signals are triggered by the comparison between the indicators value (obtained with the company information) for every day of the trading period and the threshold value selected to maximize return.

Consequently, the difficulty of solving the investment problem defined above depends on the number of indicators included in the trading system and the range allowed for the parameters to be used as threshold values for these indicators. Notice that we face a combinatorial optimization hard problem that is growing exponentially with both indicators and parameters range, being explosive when using a high number or range for both variables.

Return

To compute the return generated by the investments made following trading signals given by the indicators and parameters we use the Accumulated Return (AR_f) for the period computed as follows:

$$AR_f = \prod_{f}^{i=1}(1 + DR_i) \tag{2.1}$$

Where AR_f is the accumulated return at the end of the trading period and DRi is the daily return given by:

$$DR_i = \begin{cases} \frac{P_i - P_{i-1}}{P_{i-1}} & \text{if the TS gives a long signal} \\ \frac{P_i - P_{i-1}}{P_{i-1}} & \text{if the TS signal is short selling} \\ RFDR_i & \text{if the TS signal is neutral} \end{cases} \tag{2.2}$$

P_i denotes the stock price at day "i", while $RFDR_i$ is the risk-free daily return given by the US Treasury Bills, and TS stands for Trading System.

2.4.2 Technical analysis

Technical analysis maintains that all information is reflected already in the stock price. Trends 'are your friend' and sentiment changes predate and predict trend changes. Investors' emotional responses to price movements lead to recognizable price chart patterns. Technical analysis does not care what the 'value' of a stock is. Their price predictions are only extrapolations from historical price patterns.

Technicians using charts search for archetypal price chart patterns, such as the well-known head and shoulders or double top/bottom reversal patterns, study technical indicators, moving averages, and look for forms such as lines of support, resistance, channels, and more obscure formations such as flags, pennants, balance days and cup and handle patterns.

Technical analysts also widely use market indicators of many sorts, some of which are mathematical transformations of price, often including up and down volume, advance/decline data

and other inputs. These indicators are used to help assess whether an asset is trending, and if it is, the probability of its direction and of continuation. Technicians also look for relationships between price/volume indices and market indicators. Examples include the relative strength index, and Moving Average Convergence - Divergence (MACD). Other avenues of study include correlations between changes in options (implied volatility) and put/call ratios with price. Also important are sentiment indicators such as Put/Call ratios, bull/bear ratios, short interest, Implied Volatility, etc.

Figure 2.5: *Example of a technical indicator MACD*

Figure 2.5 represents the prices of a company in a specific period time. The black line shows the exact values of the price, the red and blue lines represent two moving averages of different term. The blue line is the moving average of short-term and the red line is the moving average of long-term. Mainly, the technical indicator MACD is build with the intersections of the two moving averages, when the short-term moving average crosses in the upward direction the long-term moving average , there is a buy signal. When the short-term moving average crosses in the downward direction the long-term moving average , there is

Table 2.2: *Formulation of indicators used for technical analysis*

Indicators representing Company Price Multiples		
Name	Description	Formula
RSI	Relative Strength Index	100- $\frac{100}{1+RS}$
MA	Moving Average	$\frac{PRCCM_t+PRCCM_{t-1}+PRCCM_{t-2}+...+PRCCM_{t-N}}{N}$
SR	Support Level Resistance Level	Local Minimum PRCCM for the last N periods Local Maximun PRCCM for the last N periods
PV	Price Volume	$\begin{cases} 1 & \text{if } PRCCM_t > PRCCM_{t-1} \& VOLM_t > VOLM_{t-1} \\ -1 & \text{if } PRCCM_t < PRCCM_{t-1} \& VOLM_t > VOLM_{t-1} \\ 0 & \text{otherwise} \end{cases}$

a sell signal.

The module is based exactly on the same structure that the one we used for fundamental analysis. The GA chooses among a set of 4 technical indicators (crossing moving average rule, relative strength index divergences, support- resistance levels, and volume indicator) the weights of the indicators to be included in the trading system as well as their parameters. Again we carry out the model in two steps, the first in which we include all the technical indicator 2.2 equally weighted in the systems, and the second step in which the GA looks also for the optimal weight of every indicator.

We use the same fitness function that we applied in the fundamental module, based on the accumulated return at the end of the training period obtained by the different technical trading systems. Our chromosomes have also the same structure that in the previous case (fundamental module), one binary chain for the approach used in step one, and a double binary chain for the one used in step two. The number of genes to codify the weights is the same that previously and the one to codify the parameters depends on the range allowed

for them according to the practice.

Chapter 3

Parallelization with a grid system

More and more personal computers are conected to the internet, the power accumulated by linking millions of computers for the same task can be impressing. That will give you a glimpse of the quake taking place in the computer industry these days. Cooperative computing takes advantage of widespread broadband connections and new concepts such as Grid, peer-to-peer and ASPs to bring the Internet to its next level.

Some applications require such high levels of computing power that they require the use of expensive supercomputers. These applications include big science (astronomy and physics), finance and biochemestry. Industry also needs more and more computing power as it shifts from real world experiments to simulation, whether for designing aircraft or for assessing a cars safety through virtual crash tests. All these applications require intensive computer resources.

Many sectors as science, medical research or business are using the grid computing. For example, Stanford University is managing a program aimed at studying genoms and protein synthesis.

A high amount of business sectors require enormous computer power, including finance, which we are interested. Buying supercomputers requires heavy investment that can be avoided by setting up computer grids. Take the example of a bank. In order to carry out its complex financial operations, it will be able to use the idle time of computers on its Local Area Network (LAN). This solution has many advantages. First, it is relatively

cheap. Second, it is scalable. If the bank needs more computing power, it will only have to tighten its grid by adding more computers to it. Some companies already provide these sevices, thus these allow hiring his services when another company require it, for instance the enterprise *Grid Systems* (34).

3.1 Introducction to *BOINC*

BOINC is an open software system for non comercial use. *BOINC* is the acromyc for Berkeley Open Infractucture for Network Computing, it was developed with the main intent of achive a massive computing capacity. This feature carry out with the interconexion of computers trought ethernet, either LAN or WAN, like a grid system.
The projects vinculated to *BOINC* has a hight exigence in capacity of computing as we can see, for example, in Seti (17) (Search for ExtraTerrestrial Intelligence), where according to SETI officials, there are currently three million PC computers participating in the program, which have the processing power of a 15 teraflop machine. Yet the total cost of the program does not exceed 500000. By comparison, IBMs ASCI White, the most powerful computer today, has a capacity of only 12 teraflops and costs 110 million.

The general architecture of *BOINC* is a client-server model, a normal flow of work starts with the sending of works units from server to clients computers. Later, when the clients has completed the work, these reported the work to server. Finally all results are processed and united in the server.
BOINC was created for exploit the lost cycles of a processor unit, that is to say, these cycles free without tasks for execute in the processor. *BOINC* support several operating systems (Unix, Windows or Mac) and several ways for execute the programs, in CPUs or GPUs. In this way, it allows interconects a set of voluntaries computers. People interested in helping to science can joined with these projects without efford and without cost for the onwer of the personal computer (the volunteer). On the other hand, many projects dedicate specific

computers for *BOINC*.

3.2 The *BOINC* architecture

The huge computation power of *BOINC* falls in the volunteer users. Therefore the *BOINC* framework consists of two layers which operate under the client-server architecture. Once the *BOINC* software is installed on a computer, the server starts sending tasks to the client. The operations are executed in the client and the generated results are uploaded to the server.

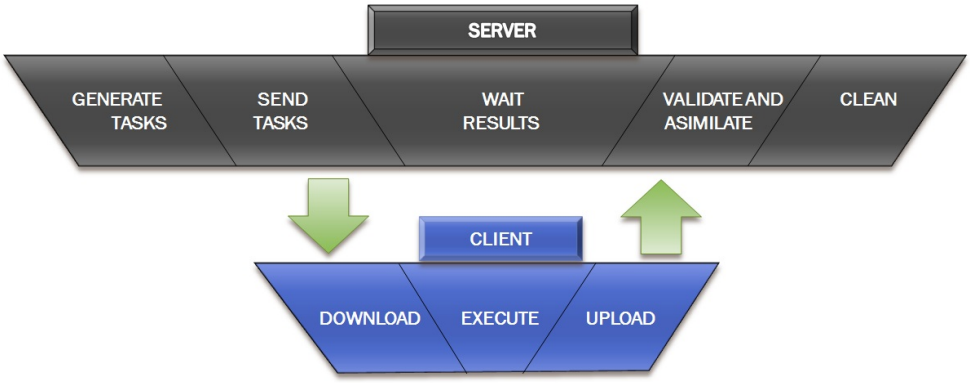


Figure 3.1: *The typical BOINC project structure*

3.3 *BOINC* server

The *BOINC* server is responsible for planning and scheduling tasks for the projects. The server interacts with all others machines, and it is the manager of sending and receiving works and results. There are several ways to install the *BOINC* server software.

3.3.1 Server like a virtual machine

BOINC provides a virtual machine to use as *BOINC* server. This *BOINC* server includes all necessary requirements to be used. These features make the *BOINC* server one of

the best options when you are starting. The operative system of the virtual machine is one distribution of Linux, specifically a basic distribution of Debian. This version do not includes a graphical interface, however you can install it later. All software of *BOINC* are provided with the original package and with all programs compiled. Furthermore this package has a ready user accounts with execution permissions. The virtual machine can be running with several software like VirtualBox (35) or VMware (36), both with free versions.

3.3.2 Independent *BOINC* server

For experimentation and debugging, you can use almost any computer as a *BOINC* server. However, when the size of the project grow, it is necessary a independent *BOINC* server for an optimal use of this software. If ours project will be executed in a huge amount of machines is highly recommended using the independent *BOINC* server. Given the fact that we have a specifically server machine, we use it for this project, thus we ensure the performance, availability, and security.

The features of our server are exposed in Table 3.1. Together with the mentioned characteristics of the server we should have an internet connection with adequate performance and a static IP address.

3.4 *BOINC* projects

When we want to set up a one application in the environment of *BOINC*, we need to create and to configure a project. A *BOINC* project is the form of naming a set of applications with a common objective and the configurations needed for to be executed in the *BOINC* platform. In the next sections we explain the different that components can be found in a typical project.

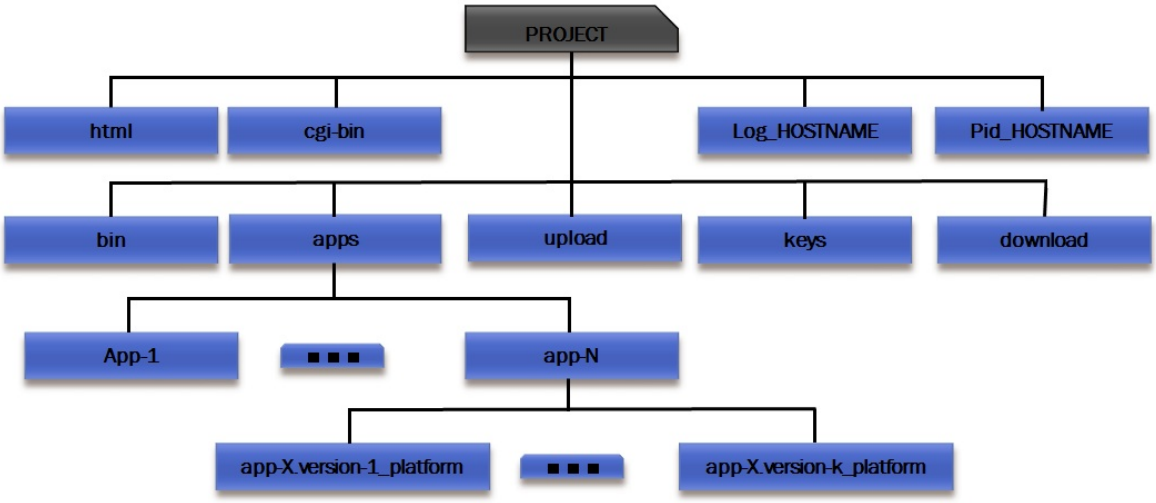


Figure 3.2: *The typical BOINC project structure*

3.4.1 Project structure

Figure 3.2 shows the general structure for *BOINC* project. This figure only represents the most important sections in the structure. The word project is the name of the project and app-X is the name of one of our applications. Version is the number of the application version, and in last place plataform is the operative sytem where the application will be executed. Each project directory contains:

- apps: Ours applications.
- bin: Server daemons and programs.
- cgi-bin: CGI (Common Gateway Interface Protocol) programs .
- download: Data for server downloads.
- html: PHP files for public and private web interfaces.
- keys: Encryption keys for our projects.
- upload: Data server uploads.

- templates: Models for inputs and ouputs.

3.4.2 Configure a project

A *BOINC* project has countless options and multiple configurations, below we talk about the different files and settings that can or should be modified for proper operation.
In the root folder of any project (see 3.2) we can find a file with the name "config.xml". This file manages a great amount of project options. For example *"homogeneous_redundancy"*: several result for the same work, *"max_wus_in_progress¨* establish a maximum of task in process at the same time, or *"one_result_per_user_per_wu¨* : sends only one result for user. Like these options, many more (see 7) can be enabled or disabled by marking the option in the file "config.xml" with a 1 or a 0 respectively.
Other function of *config.xml* is to select the daemons that our project will execute. The daemons are a sequence of programas executed while the project is active. If the tasks of a project are not finalized, the daemons of the project will continue running. These programs controls some important questions of the project, such as the management of tasks, the validation, the delivered credits. The file, contents the name of the daemons and the arguments needed for his execution, for example: *feeder –d 3*. The most important daemons are explain bellow:

- Feeder and Transitioner: These daemons are independent of the application, in other words, are part of *BOINC* and no adjustment is required, although the programs accept different commands. *BOINC* supplied this daemons, and are located in the ["bin" directory, in the root folder of the project. The Feeder creates a shared memory segment to communicate to the database the scheduler processes. On the other hand, the Transitioner is used to generate the state transitions between work units and results, generating initial results or results of error.

There are also a set of daemons which are dependent on the application, and the programmer should develop the programs:

- Assimilator: The assimilator supposes that the results have already been validated and prepares data for processing and scientific analysis.

- The WorkGenerator: This is the program that is responsible for creating and maintaining a steady stream of work until all tasks are over.

- File_deleter: This daemon works once the results have been validated and assimilated. Its function is to eliminate those redundant and unnecessary files that have been accumulated over time and have become an unnecessary burden.

- Validator: The Validator wants accurately verify the tasks have been sent to the server as completed. In this way you can avoid the "volunteer cheats" or volunteers with other purposes outside for our project.

3.4.3 Applications, versions and platforms

When speaking about a *BOINC* application we refer to a specific program inside a project. This program can actually be more for the same purpose, because of the different platforms that exist. For example, we could have an application with a version for Mac and one for Linux, or an application for Windows x64 and one for Windows x32. Moreover, each application consists of a set of work to be done. An application must have a name ("short name") that will be used to name folders and files of the application and a common name by which the volunteers know the project ("friendly name ").
When you need to make a change in any application, we must change the version number in the folder that contains your application, as well as in the main source file for the application. Thus, *BOINC* will be able to recognize the new changes in the project.
The software suggested by Berkeley have compatibility with a high number of operating systems. Nowadays, *BOINC* is a consolidated grid computing platform and brings the possibility of accessing to the higgest number of volunteer computers indeed. Therefore *BOINC* can be installed in almost all important platforms, whether with the versions of 32

bits or the latest versions of 64 bits. The more platforms supported by your project, the more users could contribute.

3.4.4 Jobs and templates

When an application is ready for execution in *BOINC*, the Server should create tasks for starting the grid computation. This tasks are commonly named as *jobs* for the *BOINC* programmers. The *BOINC* jobs has two different sections, workunits and results.
A workunit is technically a portion of the program that describes the computation to be performed. A workunit has one or more results, each of which describes an instance of a computation, either unstarted, in progress, or completed. The *BOINC* client software refers to results as "tasks".
The generation of workunits depends of the programmer. We need a script (sh script) to generate the workunits for our project. This script use two templates for the specification of the jobs, the workunit template and the result template. The files of these template are stored in the folder templates of the root directory of our project. In the work unit template we specifies the names of the input files needed for the project and in the result template we specifies the output files generated for ours applications. In this way, *BOINC* knows the inputs and outputs of ours applications.

3.5 Wrapper

Wrapper is a program provided by *BOINC*, wich is able to execute any type of application in the environment of this platform. Thus, Wrapper become to a very useful tool when the programmer cannot access to the source code, or in cases where the code is difficult for support changes, even when the code is not developed by languages supported by *BOINC*. This program encapsulates the original application, as a result *BOINC* can processe the application 3.3. One program is not the limit of Wrapper, and it can execute a sequential number of applications and even it can establish checkpoints for heavy applications.

Wrapper execute the programs like subprocess, so the communications of Wrapper with the *BOINC* environment are fluid.

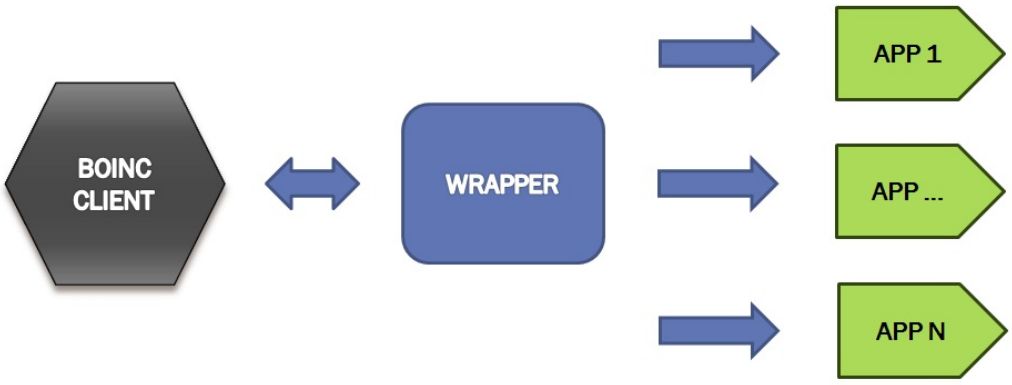


Figure 3.3: *Wrapper operation*

The execution of Wrapper will be determined by a task descriptor as a XML file, the name of this is: job.xml. This file can introduce various important options for our application, like checkpoints (*Checkpoint_filename*), the arguments of our application (*command_line*), and more (see 6).

3.5.1 *BOINC* database

The server *BOINC* database is a set of tables and indexes that contains the information stored for all projects in the *BOINC* server. *BOINC* stores the data in a MySQL database. We can find tables for the workunits, results, applications, user, etc.
The database for each project is generated by the make_project script (this is a script that you should execute when you create a new project). Normally, you don't have to directly examine or manipulate the database, however occasionally is very useful to manage the workunits and results, because you can obtain statistics, delete the workunits failed, etc. There are several ways to access to the database; you can use the MySQL command-line interpreter or the *BOINC*'s administrative web interface.

3.5.2 *BOINC* client: Inputs and outputs

In order to develop applications with *BOINC* is important that we know the *BOINC* client. *BOINC* client is a light program that remains in communication with the server. The most important subject that we must know is the structure for the inputs and outputs, in general for debuging.

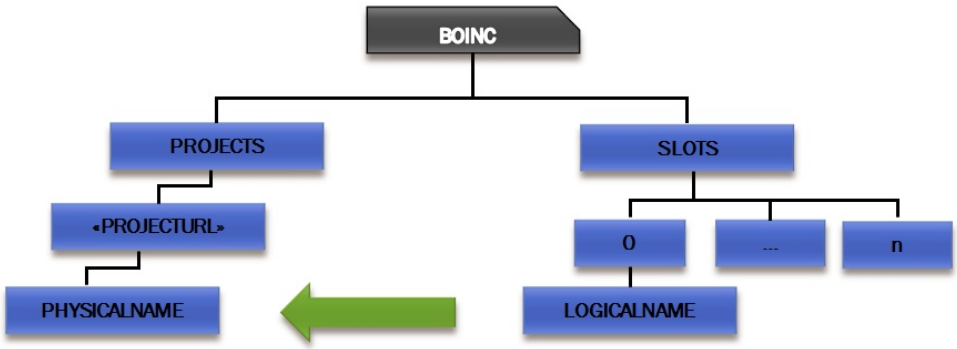


Figure 3.4: *Inputs and outputs BOINC client structure*

In the client computer each *BOINC* project has the structure represented in the Figure 3.4. Each project has a specific directory, and in them are store all data connected to the project. These files correspond with the *physical names*. Every task executed will have a different folder and these will reside in the directory *slots*. Each one of these folder has a number for his identification, for example if the *BOINC* client has 8 tasks, the directory *slots* will have 8 folders with names 0, 1, 2, and so on. Furthermore, this directory will have the links to the inputs and outputs of the tasks, these names are named *logic names*. The previous configuration allows to link the *physical names* by one function of the *BOINC* API (*boinc_ resolve_ filename()*). Thus, *BOINC* achieves two fundamental advantages. The first is that a lot of tasks have access to a file without hold different copies of this. And the second one, the applications can link the logic files at the same time that they are executing several physical files.

3.6 Modifications in the code

The original code is an unique program that takes as input a matrix with the data of companies in the S&P 500 for a period of 1986 to 2006. This program executes the genetic algorithm several times, one for each company. As a request, the genetic algorithm is executed approximately 4500 times. The way for parallelize the program is to divide these thousands of executions in independent tasks. Thus, we can execute the genetic algorithm full, without partitions, this means less problems and less complexity for the parallelization of the code. Of course, 4500 tasks are enough for achieving one of the best configurations with our grid. The task are enough big for not overload the server with a lot of requests, and are enough little for not overload the computers with long executions.

Thanks to this way of dividision the program, the modifications needed in the code for achieve a parallelized program are small. Mainly, we need to change the principal file where the execution of the GA is processed and create a script for reading the data, executing the algorithm and saving the data. In this order and with the correct parameters.

The script executes the program in the adequate order, it is a file written in *Matlab* programing language. This script receive an unique string with the year and the company number (acording to the data that the program runs). First the script reads data from the company to be stored in an array. Second, it executes the algorithm with the default parameters, as the number of individuals, generations, percent of mutation, etc. And third, we need to save the output in a file with the same name of the data of the company.

On the other hand, we need to do modifications in the main file of the genetic algorithm. Thus we changed all code for processing the data in two dimensions and lonely for one company.

3.7 Parallelization tasks

3.7.1 Input pre-process and output post-process

The original code has an Excel book as input. This Excel file is divided by years, the pages of the book, and by companies, all on the same page. These files have a weight between 20 MBytes and 80 MBytes, so if the *BOINC* server sends the full input to the client computers, we will overload the grid system. This overload is unnecessary because we use only one company in the execution of the program in a grid node. Then we need to divide the input data into independent data for each company. Thus the program will run with the minimum amount of data. We have created a little program that divides the original data in a collection of Excel files. Finally we have around 10000 files; 5000 for technical analysis and the same number for the fundamental analysis.

At the same time, when all tasks are ended, we have a great amount of files with the final results (one result for each task). We need to develop a program that combines all the results in the same once more. The results are stored in a ".mat" file with several matrices and variables. These matrices and variables represents the final result of our genetic algorithm, then we take all the results and create a collection of matrices and variables with another dimension. With the pre-process we achieve all the results in one file, and a grid with more fluid communications.

Matlab in the client

The first option to consider is to install a version of *Matlab* in the client. In this way the source program must be a script (sh script) that executes *Matlab* in the client. The input of this script should be the name of the main file of our program. The main file will be created like an executable, this proccess will be explained in section C. Is highly recommended that execution runs without graphical interface, thus the user (computer where the *BOINC* client is installed) has not knowledge of his execution.

If we follow this way, all files needed for our applications must be added to the input

template, including the main file of our program, because the file executed by wrapper is the script mentioned above.

It is appropriate to mention the two ways to transmit the commands in *Matlab*. The first with indirection operator "<":

```
./matlab -nodisplay -nosplash -nodesktop < mainFile.m
```

Or with a specific command (the file ".m" without the extension)

```
./matlab -nodisplay -nosplash -nodesktop -r mainFile
```

This last option has a big problem, because all the clients must have installed the *Matlab* program, which limited the proportion of volunteer users. Moreover, the software is a program with a not free license that limited again the proportion of potential volunteers. Even in the case where the client has the *Matlab* software, the execution may throw exceptions by incompatibilities between the different versions of the software.

Matlab executable and MCR

The second and more desirable option is to use a *Matlab* Executable C encapsulated by Wrapper. *Matlab* has the ability to convert a program written in his language to a single executable file that will depend on the target platform.

For the right operation of the executable file, the client computer should have installed MCR (*Matlab* component runtime). MCR can run almost all *Matlab* functions, including all functions derivatives of these toolboxes. The MCR may be freely distributed with the library files generated by the *Matlab* compiler. In the case of Linux, furthermore of the executable, we will create a script (sh script). This script will be necessary for the execution of our application. This script has the functionality of enabling the libraries of the MCR environment and running our application.

It is also possible to package the application together with MCR libraries. In this way we would have a single executable which brings together all the elements needed to run the application on any computer. . The MCR libraries are distributed by *Matlab* and we can

find the MCR on the PC where the software is installed *Matlab*:

```
MATLAB/"version"/toolbox/compiler/deploy/win32/MCRInstaller.exe
```

Something to keep in mind is that the version of the MCR must be the same or later to the version of *Matlab* where we implemented the code. In other case, the MCR and the code could be incompatibles.

3.7.2 Configuration

The *BOINC* platform is a very powerful system, however will need a good knowledge in IT if we want take advantage of *BOINC*. Since the installation of the *BOINC* server until the execution of the project is needed the configuration of different software and the monitoring of specific steps.

- First, We compiled the source code of Wrapper for Linux and windows with a C compiler. The code of Wrapper and the linked libraries are located in the next directory:

  ```
  boinc/samples
  ```

- Later, we created the structure of the files and folders for our project and applications.

- We change the files project.xml and config.xml with specific data of our project. Furthermore, in these we have activated a collection of options for facilitate the work of the projects, like the *homogeneus_redundancy*.

- The default web interface has been activated and modified for ours proposes.

- We have implemented a work generator for the creation of task. We developed a sh script that generate a collection of templates for all workunits and results of all runs of our program.

- And finally, the file job.xlm linked with wrapper has been modified with our data and we have activated some options like "fraction_done_filename".

3.7.3 *Matlab* and *BOINC*

Matlab is not compatible with the *BOINC* API, so we have to use the encapsulator program Wrapper, explained above. On the other hand, *Matlab* uses a lot of libraries to run their own code; *Matlab* also has the possibility of using Toolbox calls that require more additional libraries. Therefore, we cannot run programs written in *Matlab* in the *BOINC* platform without previous preparation by the customer and/or the source program.

3.8 Experimental results

In the follow sections we will present a description of the experimental tests that we have done with the program in the *BOINC* enviroment.

3.8.1 Metrics

The set of experimental tests have been carry out in a *BOINC* grid installed in CES Felipe II *(A Computer University College of Aranjuez, Madrid, Spain)*, thus we have used the computers of different laboratories and some well-known volunteers *(falua.cesfelipesegundo.com)*. Tests are used for estimating the grid computing power, because the capability to support volunteers carries a nonconstant computing power. These tests consist of a series of computing time measurements in a independent CPU and in the computing grid. The execution times of the grid tests were obtained with the administrator page of the *BOINC* project. Moreover, the execution times of the basics executions in the CPU were measured with the *Matlab* Profiler tool in a computer with a Pentium 4 processor (see 3.2). *BOINC* client and the MCR libraries of *Matlab* was installed in all computers. The trading system has been executed fully, for all companies and for all years, which means approximately 5000 runs of the GA.

In Table 3.1 we can see the mainly characteristics of the server. The IBM computer is a professional server, specially designed for this purpose.

Table 3.1: *Main characteristics of the Server used in the experiments*

Server	IBM xSeries 236 Type 8841
Processor	Intel Xeon 2,8 GHz
RAM	2 Gb
HDD	x4 70Gb - Raid 5

Table 3.2: *Main characteristics of the grid used in the experiments*

Group	**PCs**	**CPU**	**Operative System**	GFLOPS	GIPS	**Total** GFLOPS	**Total** GIPS
Lab. ITIS 1	20	2 x Intel P4 3GHz	Windows XP x86	2,744	5,101	**54,88**	**102,02**
Lab. ITIS 2	21	2x Intel P4 3GHz	Windows XP x86	2,744	5,194	**57,624**	**109,074**
Lab. ITIS 3	22	2x Intel P4 3GHz	Windows XP x86	2,744	5,01	**60,368**	**111,22**
Lab. I4	1	2x Intel E2200 2GHz	Ubuntu Linux x86	1,853	5,436	**5,905**	**13,204**
	2	2x Intel P4 3GHz	Ubuntu Linux x86	2,026	3,884		
Lab. DOSI I+D	2	2x AMD Athlon 4600+	Windows XP x86	4,906	8,974	**11,84**	**21,556**
	1	2x Intel P4 3GHz	Ubuntu Linux x86	2,028	3,608		
Volunteers	6	2x Intel P4 3GHz	XP x86 XP x86	2,722	4,747	**35,128**	**77,53**
	2	2x Intel P4 3GHz	Ubuntu Linux x86	1,847	2,876		
	1	4x Intel i5 750 2.7GHz	Windows 7 x64	11,492	36,736		
	1	AMD Athlon 2600+	Windows XP x86	2,129	3,578		
	1	2x Intel T2450 2GHz	Windows XP x86	0,802	1,508		
	1	Pentium III Coppermine	Ubuntu Linux x86	0,679	1,474		
TOTAL						**225,745**	**433,604**

Table 3.2 summarizes the main characteristics of the different groups of computers that comprise the system (processor type and operating system) and the measures undertaken to estimate the computing capacity of the system.
We have carried out two tests of performance in each of the computers for determine both magnitudes (GFLOPS and GMIPS). The name of the test are *Whetstone* and *Dhrystone* provided by the *BOINC* (These can be found in the *BOINC* client). Once done, it is taken the maximum value of the same for each group of computers with the same processor and operating system. In short, the grid Falua has about 225 GFLOPS and 433 GMIPS. These numbers supposed the grid at full capacity, with all the computers active and available.

Others software packages are used for managing the *BOINC* grid. We need to manage all computers (no volunteers) in our grid, because is very uncomfortable to manipulate the computers independently. We used the software EMCO Remote Shutdown (13), to a great extent for to turn on/off ours computers. We also need a manager for the *BOINC* client, it is heavy and inefficient going computer by computer, for example for join the computer in a project, or request more tasks. For this purpose we used the BoincView 8 that facilitates the uses of a lot of computers with the *BOINC* client, that allows you to manage the *BOINC* Client on a single PC.

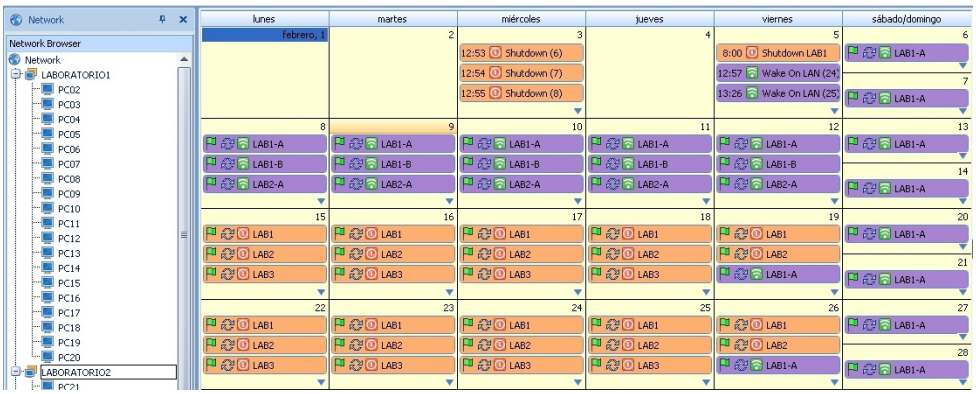
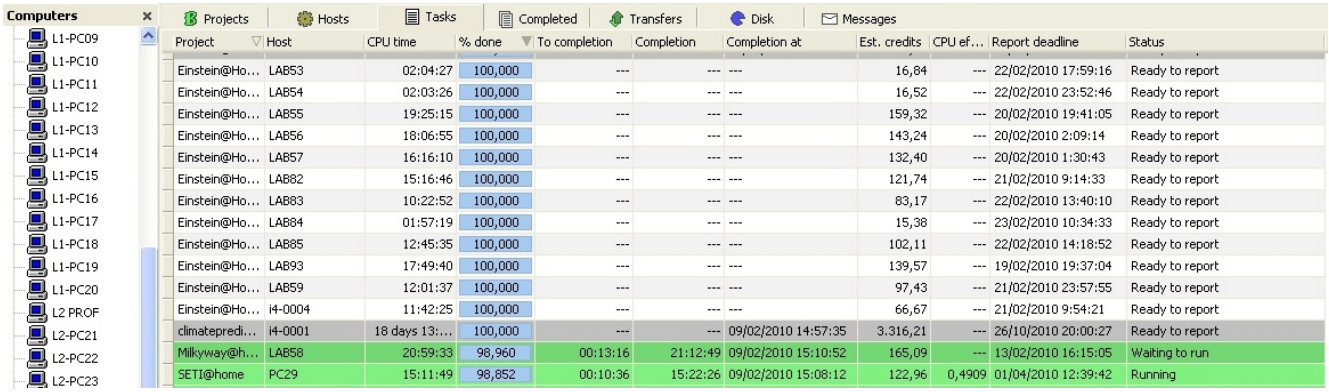


Figure 3.5: *Detail of EMCO Remote Shutdown screenshot*

Figure 3.5 shows the main interface of the EMCO Remote Shutdown, we can observe the managed tasks We Figure 3.6 BoincView software



Project	Host	CPU time	% done	To completion	Completion	Completion at	Est. credits	CPU ef...	Report deadline	Status
Einstein@Ho...	LAB53	02:04:27	100,000	---	---	---	16,84	---	22/02/2010 17:59:16	Ready to report
Einstein@Ho...	LAB54	02:03:26	100,000	---	---	---	16,52	---	22/02/2010 23:52:46	Ready to report
Einstein@Ho...	LAB55	19:25:15	100,000	---	---	---	159,32	---	20/02/2010 19:41:05	Ready to report
Einstein@Ho...	LAB56	18:06:55	100,000	---	---	---	143,24	---	20/02/2010 2:09:14	Ready to report
Einstein@Ho...	LAB57	16:16:10	100,000	---	---	---	132,40	---	20/02/2010 1:30:43	Ready to report
Einstein@Ho...	LAB82	15:16:46	100,000	---	---	---	121,74	---	21/02/2010 9:14:33	Ready to report
Einstein@Ho...	LAB83	10:22:52	100,000	---	---	---	83,17	---	22/02/2010 13:40:10	Ready to report
Einstein@Ho...	LAB84	01:57:19	100,000	---	---	---	15,38	---	23/02/2010 10:34:33	Ready to report
Einstein@Ho...	LAB85	12:45:35	100,000	---	---	---	102,11	---	22/02/2010 14:18:52	Ready to report
Einstein@Ho...	LAB93	17:49:40	100,000	---	---	---	139,57	---	19/02/2010 19:37:04	Ready to report
Einstein@Ho...	LAB59	12:01:37	100,000	---	---	---	97,43	---	21/02/2010 23:57:55	Ready to report
Einstein@Ho...	i4-0004	11:42:25	100,000	---	---	---	66,67	---	21/02/2010 9:54:21	Ready to report
climatepredi...	i4-0001	18 days 13:...	100,000	---	---	09/02/2010 14:57:35	3.316,21	---	26/10/2010 20:00:27	Ready to report
Milkyway@h...	LAB58	20:59:33	98,960	00:13:16	21:12:49	09/02/2010 15:10:52	165,09	---	13/02/2010 16:15:05	Waiting to run
SETI@home	PC29	15:11:49	98,852	00:10:36	15:22:26	09/02/2010 15:08:12	122,96	0,4909	01/04/2010 12:39:42	Running

Figure 3.6: *Detail of BOINC Viewer screenshot*

3.8.2 Execution Time analysis of the algorithm in the grid

Figure 3.7 shows the related data with a set of executions where the parameter used to determine the number of individuals iteratively get more weight. These series of executions have been carried out with 500 generations. The number of individuals is represented on the x-axis. The increase intervals in this parameter are not identical along the graphic. The first five intervals raise the amount of 100 individuals each one, and the next 10 intervals raise with 1000 individuals each. On the other hand, the y-axis shows the execution time for the algorithm, and is represented in a 10-base logarithm scale.

From the beginning, the implemented application in the grid starts to provide beneficial execution times. Technical Analysis is heavier than Fundamental Analysis, this uses huge amounts of data indeed, for this reason it achieves the highest times. We can observe the inefficient times reached in an independent CPU. For example, we can analyze the first bars of Figure 3.7, where the times in technical analysis for 500 generations and 500 individuals are bigger than 100 days (184,9 days) and approximately one week in the grid version (4,11 days). And the same for fundamental analysis where the grid version achieve the results in

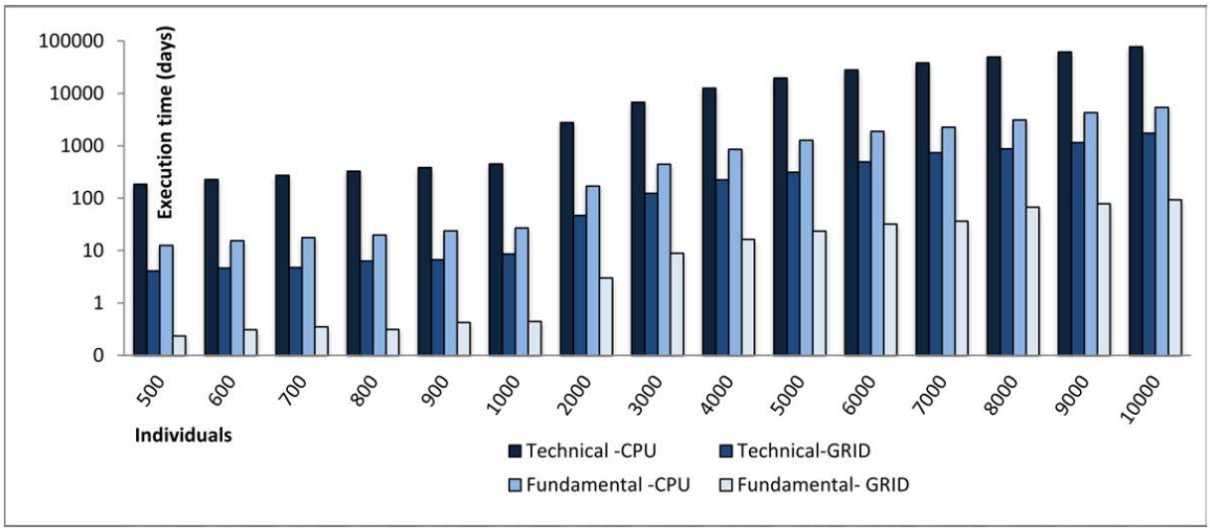


Figure 3.7: *Execution times in the grid (y- axis) for 500 generations and different number of individuals (x- axis)*

a few hours (5,6 hours) while one computer spends about 10 days (12,64 days). 100 days of execution is a very high time, fully uneasy, even more if we thinks that the execution depends only on one machine, with an execution without checkpoints. In this way, the system becomes easily susceptible to any danger or event, like one failure of the software or a cut of energy.

Figure 3.8 represents the power of the grid respect to the independent computer version, in other words, we represent the speed-up of the grid. We can observe that numbers are approximately constants. The achieved speed-up is around 50 units and this ratio remains throughout all tests realized. The speed-up is not continue at all, this is because the computers in the grid not have a continued availability, maybe one computer is off by hours or it have a failure or another event out of our control.

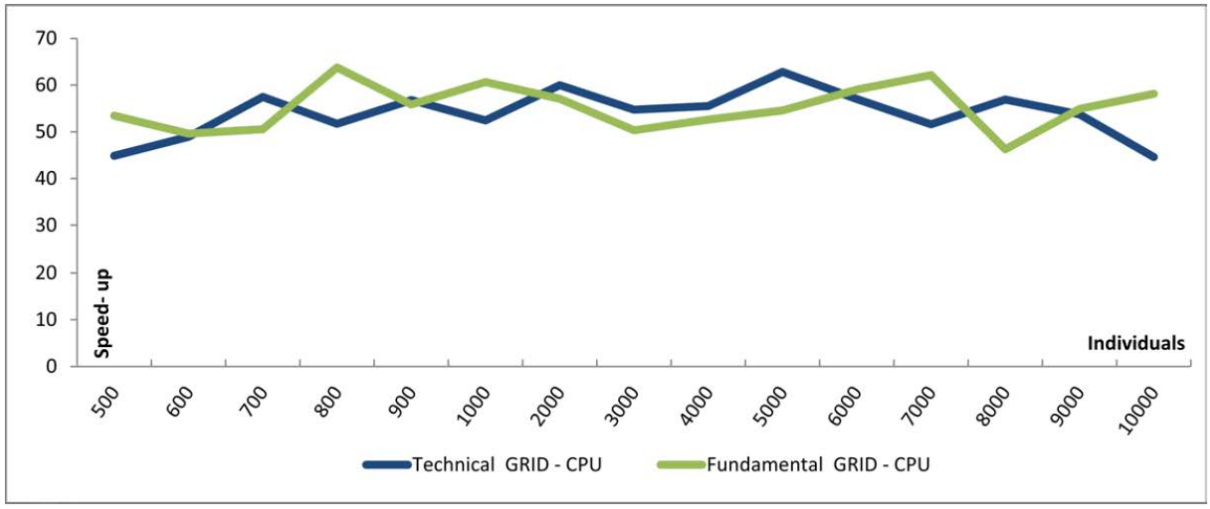


Figure 3.8: *Speed-up in the grid*

Chapter 4

Parallelization with a computer graphic card

Until now we used a grid system, however we have not been able to rise the amount of individuals a lot. From this point, we will search another way from take advantages in the computation time. The developing of our program in a GPU.

We can find in the literature several approximations for implementing evolutionary algorithms on GPUs (22; 20; 30; 4; 25; 37). Most of them rely on the CUDA architecture (29) and provide detailed information on how to configure the control parameters in order to obtain an efficient implementation. Those works show that doing an ad-hoc implementation require a good level of knowledge on a set of computer architecture and programming issues. However, our proposal tries to offer an adaptable tool for investor with no special knowledge on computer architecture, although familiar with *Matlab* (23) tools. In this way, we proposed an implementation based on a software tool named *Jacket* by AceelerEyes (1). In this chapter we will explain the general structure in a graphics device, the motivation for this selection and several parallelization and implementation details.

4.1 CUDA architecture

Multithreading in processors of general purpose is a common technique. This technique are used for take full advantage of available resources. The processor in collaboration with the

operating system can process instructions of two or more threads simultaneously. Thus, taking advantage in the parallelism in a level of thread.

In the tasks that are designed for the graphics processors, the parallel is easily exploitable. There are calculations to be performed for each vertex or each fragment, which means repeating the same task over and over again on different data in memory, so the idea of parallelism and multithreading is essential in the design of current GPUs.

CPUs devote a lot of transistors for cache memory and for control flow. Instead, a GPU uses most of the area for the ALUs. The memory accesses of a CPU and thread changes are much slower than the access to memory made by a GPU. In addition GPUs have are designed to execute in parallel large numbers of threads. However, not all are advantages in GPU programming. Programming languages for GPU are usually at low level and it is necessary to use a graphics API for converting data to images, or for converting any type of algorithms in a image processing algorithms. In short, the learning curve in these languages is very slow. Moreover, it should be noted that GPUs have a limit in the memory bandwidth, as they consume a lot of the available on the exchange of information. To fill gaps in these GPU languages, CUDA was born as a general purpose language and parallel software development environment. CUDA is a step to facilitate the developing for the programmers in the GPU implementation. CUDA uses a version of C instead a graphics API.

Figure 4.1 show the general architecture GPU-CPU. The arrows represents the connections between differents modules. There are two orange arrows that represent the last two CPU Intel architectures (18),FSB (Front Side Bus) and QPI (Quick Path Interconnect). FSB links the DRAM memory through the northbridge, and QPI integrates a controler in the own processor, thus allow the direct link with the memory DRAM.

Now, we will show some figures for the understanding the architecture of a modern GPU, specifically the Nvidia GTX 280 (compatible with CUDA).

Figure 4.2 depicts a high-level view of the GeForce GTX 280 GPU parallel computing

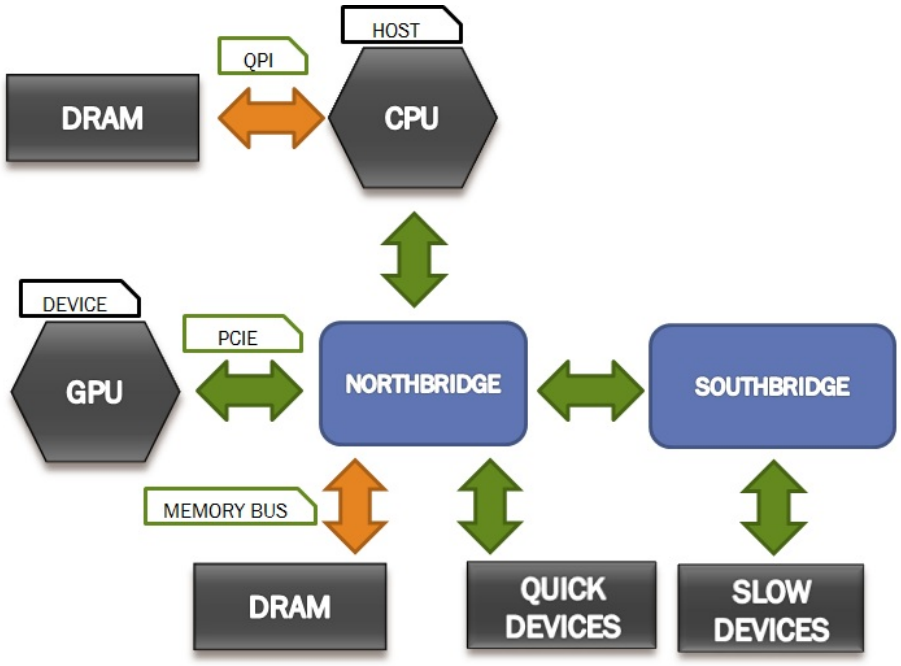


Figure 4.1: *General architecture CPU-GPU*

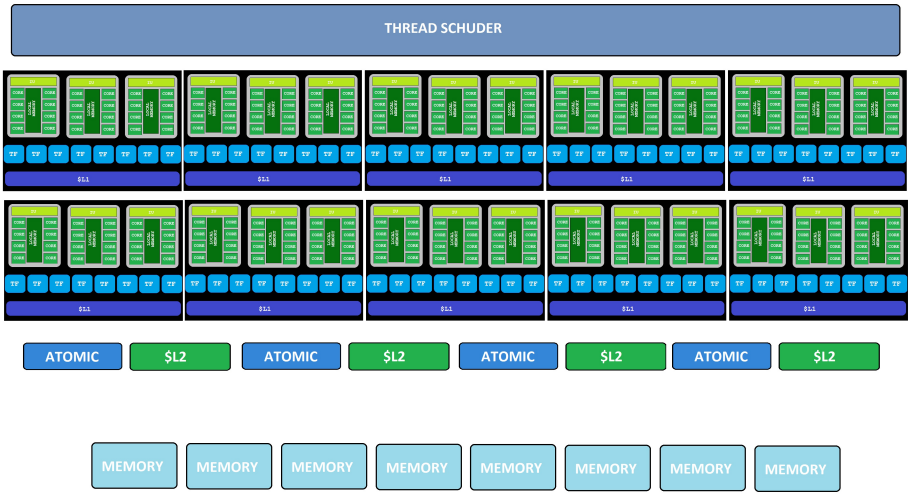


Figure 4.2: *GeForce GTX 280 GPU Parallel Computing Architecture*

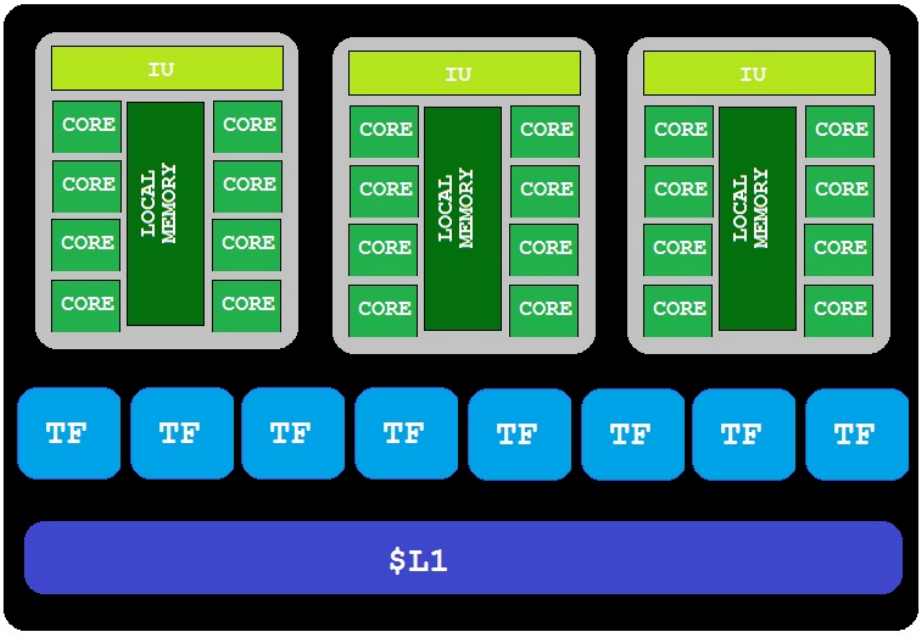


Figure 4.3: *Detail of a Thread Procesing Cluster (TPC)*

architecture. A hardware-based thread scheduler at the top manages scheduling threads across the Thread Processing clusters (TPCs). Furthermore, we have operatives the texture caches and memory interface units. The texture caches are used to combine memory accesses for more efficient and higher bandwidth memory read/write operations. The elements indicated as "atomic" refer to the ability to perform atomic read-modify-write operations to memory. Atomic access provides granular access to memory locations and facilitates parallel reductions and parallel data structure management.

A Thread Procesing Cluster in compute mode is represented in Figure 4.3 below. Each TPC is in turn made up of a number of streaming multiprocessors (SMs), and each streaming multiprocessors contains eight processor cores (SPs). You can see local shared memory is included in each of the three SMs. Each processing core in an SM can share data with other processing cores in the SM via the shared memory, without having to read or write to or from an external memory subsystem. This contributes greatly to increased computational speed and efficiency for a variety of algorithms.

As we can observe, despite the facilities that CUDA provides to programmers, CUDA is a difficult enviroment for people who rarely developed software. For that, we propose to use a more easy tool, the *Jacket* software (see 4.1.2).

4.1.1 Execution time analysis in CPU

Being a genetic algorithm, the main program structure is a *for loop* with a certain number of generations. Parallelizing directly the execution of this *for* in a lot of threads is unfeasible, since this would prevent the populations evolution and the concept of genetic algorithm would lose its meaning. To parallelize these cycles, we would have to change the basic structure of the algorithm as other approximations, such as island model, usually do (10). Having in mind the structure of the GA, it is necessary to look for a parallelization in its basic operators: selection, evaluation, crossover, etc. In order to determine the critical elements a time analysis for the different processes that form the main program has been done. To analyze the program execution time, a *Matlab* profiler (23) has been used, with which the different execution time used by each function can be distinguished. Thanks to these time measurements the parallelism of the most controversial areas can be influenced in so far as to computation time.

Figure 4.4 shows the 15 functions with greater weight in the GA computation time. Their names are listed in the base of each column. For example, "*main*"is the main program and "geneticAlg"is the main genetic loop. Some function names are marked with an asterisk; all these functions are MEX functions. These functions have been written in C language, and are used to manage external libraries, in this case the *sortrows* function. *Total Time* is the time that the program is working, as it can be noticed *main* is active practically all the time during the program execution as expected. *Self time* is the total time minus the time that shares with the calls to other functions. The *repopulation* file is the part of the GA code which evaluates the random population and selects a new population for a further crossover. As it can be observed in the graphic, *repopulation* spends part of the

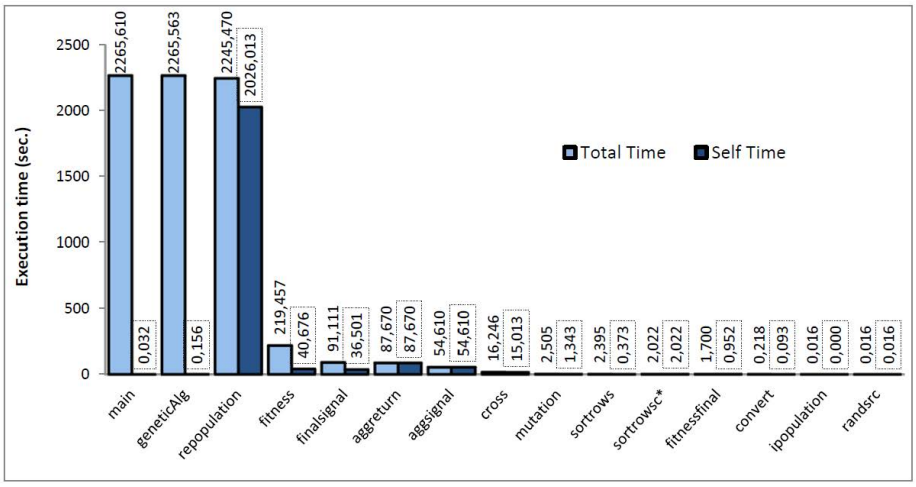


Figure 4.4: *Genetic algorithm execution in the CPU. It has been run for a company with 5000 individuals; 500 generations and the roulette wheel selection algorithm. Total time = 2266 seconds.*

time running in other files that correspond to population assessment (i.e. *fitness*) and other time running a selection algorithm directly embedded in *repopulation*. Knowing the division of this file and analyzing the graphic we can conclude that most of the time is devoted to run the selection algorithm, taking almost 90% of the execution time. One important point to remark is that the initial implementation used roulette wheel selection, widely used in genetic algorithms although computationally expensive.

As we can check, the main limitation of the efficiency of the program was found in the selection algorithm, and is here where we should focus the parallelization in order to reduce the execution time of the algorithm. However, we not only will affect the parallelism of the selection but also will try to optimize the cost of GPU-CPU context switching. So, in order to make more profitable the parallelization in the GPU, we shall use during the main loop the data located in the graphic card and run the greater part of the program in the GPU. As we have mentioned, we take profit of *Jacket* capabilities to perform the parallelization of the code.

4.1.2 *Jacket*

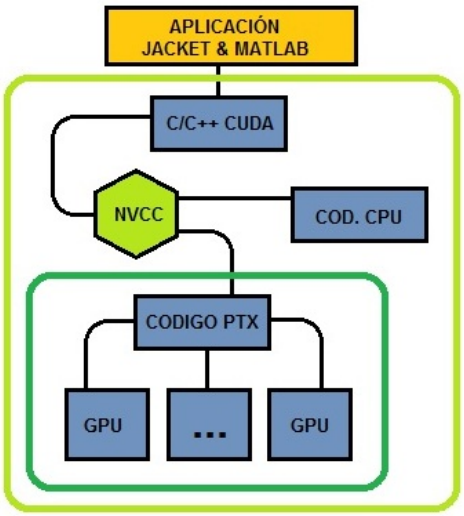


Figure 4.5: *Typical genetic algorithm work flow*

Jacket is a numerical calculation software solution developed by *AccelerEyes*. The choice of this software is motivated by several characteristics of the tool listed below:

- The main characteristic of this tool lies in its ability to accelerate codes based on *Matlab* by a GPU.
- *Jacket* offers the possibility of manipulating matrices easily in the GPU.
- The implementation includes interfaces with other popular programming languages such as C, C++, and CUDA.
- *Jacket* provides a set of GPU versions of most of *Matlab* functions, which facilitates programming if compared to any programming language for CPU. A good example is the generation of random numbers, while in most of the GPU languages are not immediate and can generate problems (21), in *Jacket* it is as simple as the use of a single command. Random numbers are a key factor in the correct functioning of the evolutionary process.

- *Jacket* introduces new specific data types from GPU in *Matlab*. Once a GPU data structure has been created, any operation in which this matrix is used will be implemented in the GPU instead of the CPU. In order to stop the GPU computation, we simply must pass the data to the CPU using one of the data types from *Matlab* (*double*, for instance). *Jacket* includes a graphic library, which completes its main characteristic, because with this library computing visualizations in the CPU can be generated. Its use is based on commands, that are versions of the visualization common commands of *Matlab*, as for example *gplot* o *gsurf*.

Thus, *Jacket* gives us the enough capability to compute *Matlab* program with no specific knowledge of the structure of the GPU, which is highly desirable in order to create a useful investment tool. Remember that the final user of the methodology probably will not be a computer science expert.

4.1.3 Parallelization Tasks

We have performed a set of parallelization tasks in order to adapt the code to the GPU:

- The first point was to make a pre-localization of the data inside the memory is performed, since according to what was written in the code, arrays change their sizes with the advancing of the execution of code.

- Then a genetic casting of the data loaded by the CPU to the data of the GPU, this should be made before entering the main genetic algorithm loop.

- Functions for generation of random numbers and function of array creation are substituted by their equivalent in programming with *Jacket*.

- Finally, all the *for* loops included in the selection, evaluation, crossover and mutation procedures are parallelized.

Special attention should be paid to the selection algorithm. The roulette algorithm has a computing cost of quadratic order, since it consists of two nested loops. *Jacket* provides a

large capacity of parallelization with *for* loops, thanks to the *gfor* command. Gfor command allows to execute in parallel the different iterations of a *for* loop, and although the basic operation is the same as a normal *for*, we must be very careful when using it because *gfor* carries many incompatibilities. Relevant documentation on this command points to several inconsistencies, some of them are important to mention owed to its direct involvement with the problem analyzed. One of these incompatibilities is that inside a *gfor* loop is not possible to nest anymore *gfor* blocks, neither to include the *break* command nor the *if* command. Due to these inconsistencies, a need arises to change parts of the code as explained in next subsections.

After the changes made to the code, we tried to parallelize the inner loop, since the *gfor* initially is compatible with *for* o *while* loops. After several implementations and different contacts with the *Jacket* technical information service, we can confirm that although *gfor* is compatible with simple while and *for* loops, it is not compatible with more complex loops (as is in our case), and therefore the point where the problem requires more parallelism could not be solved in a sufficiently effective way to get a substantial advantage in the execution on the GPU.

With this motivation we decided to change the roulette wheel selection algorithm for another one, also classic in genetic algorithms, the tournament selection (24). This algorithm, which is able to obtain the same (or even better) quality in the results, is clearly less computationally expensive because it has a cost of lineal order. As we can see in the following graphic the executing time in CPU is small due to the change in the algorithm. Some other previous implementation of Evolutionary Algorithms on GPUs adopted similar solutions (2; 22; 21; 20).

Figure 4.6 shows the execution time for the 15 functions with the highest CPU execution time in the GA. The main difference with Figure 4.4 is the change of the selection algorithm. This change, roulette wheel by tournament selection, has led to a decrease of approximately 75% of the total execution time. Nevertheless, the reduction of the execution time in this

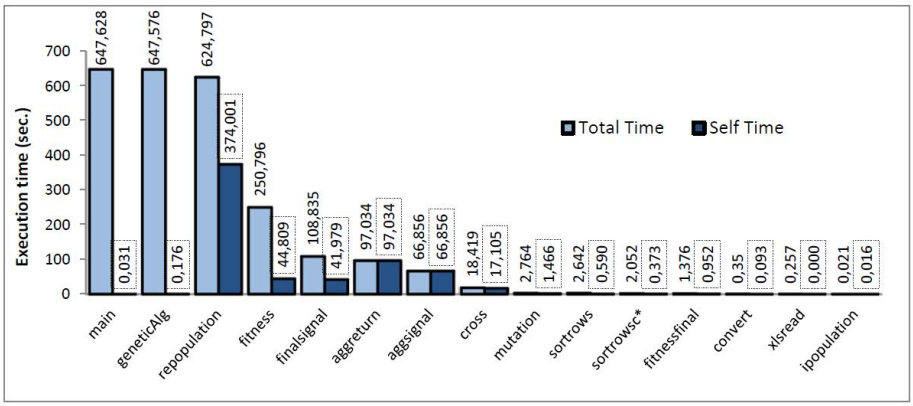


Figure 4.6: *Execution time analysis of the genetic algorithm in the CPU. (One company, 5000 individuals, 500 generations and tournament selection algorithm). Total Time = 648 seconds.*

algorithm does not reduce the time spent in the selection in proportion with the other functions used. As in Figure 4.4 could be easily deduced, the weight of the selection algorithm by the roulette method, in Figure 4.6 is reflected that the time of the selection algorithm by tournament takes more than half the time of the total execution time of the program.

4.1.4 Modifications in the code.

In order to adapt the code to the graphic card when using *Jacket*, it has been necessary to adapt some parts of the code. Reviewed below are the changes considered more important and useful for the reader.

Previously to starting with the changes associated to the parallelization of the code, a pre-assignment of the memory space has been made for all the arrays used throughout the code. This is because in *Matlab* is not necessary the declaration of variables. For this cause, we need be careful with the use of them. Looking a best compression to the problem we expose the following example. Let us suppose the following *Matlab* code fragment:

```
for  i=1:1:max
        A(i,i)=1;
end
```

In the first iteration, when *Matlab* carries out the assignment of the variable, makes a matrix of dimensions $1x1$, however in the second iteration the size of the matrix should be $2x2$, so the software needs expand the assigned memory for this variable. At each iteration, until the final of the loop, the software should realize the same operation. Furthermore, the space adjacent to the memory space of our variable may be insufficient for locating the new size for the modified variable, and the software will have to re-allocate the variable in other place. These problems could appear at each iteration. To fix the described problem, we just use the pre-allocation of the variables. Hence, we will modify the *Matlab* code in the following way:

```
A=zeros(max,max);
for  i=1:1:max
          A(i,i)=1;
end
```

Not pre-assigning the memory is inefficient and can carry out performance losses with the continuous changes in size of the arrays, especially when the sizes of theses arrays are huge. Any program designed for its execution on a CPU should be modified in order to be able to carry out its execution by a GPU. This is due to some limitations existing in the GPU programming languages and to architectural differences between both processing units. The code transformation for its execution in a GPU usually requires drastic changes, and it is here where *Jacket* shows its strong points. *Jacket* is conceived to gain efficiency in the programming execution without having to resort to large modifications. Still, while keeping the basic structure of the code, it is necessary to apply a series of changes to the original code. The magnitude of these changes will depend on the kind of parallelism we want to apply and on the code itself. The first modification that must be applied to allow the GPU programming is to locate the data on the memory of the graphic card. In our case, we have two ways of locating these data:

1. Creating the data directly on the graphic card. In that case the modification is small, as *Jacket* supports functions such as *grandones* or *gzeros*, with a performance equiv-

alent to their counterparts for CPU. These functions are very useful and can largely facilitate programming. Specially the grand function that simplifies and solves one of the difficulties in GPU programming, the random numbers. This way, has been used when the variables can be created in the memory of the GPU, and these, will have an use with other variables of GPU.

2. The second way is to perform a casting of the CPU data to the data for the GPU, for instance:

   ```
   GPU Matrix = GDOUBLE(Matrix)
   ```

   We place in this case when we need data located in the normal memory, which uses the CPU.

Either to store them or to use them in the CPU, if we want to take the data out of the graphic card, (in our case, once the main loop of the genetic algorithm is finished) a similar casting is performed, for example:

```
Matrix= double(GPU Matrix)
```

The remaining changes are linked to the incompatibilities that *Jacket* shows when programming in GPU. The most significant changes are located in those parts of the code that are included inside a *gfor* loop. Remember that a *gfor* loop is identical to a *for* loop in its general functionality, however the first causes that the iterations of the loop are executed in parallel in the GPU. Due to its characteristics, accumulators will not be able to be used inside this type of loops, because are incompatible with this command. One example is the fitness accumulation which is necessary in order to apply the roulette wheel selection method. In this case a *gfor* loop will not be able to contain branches inside, that is, it could not be possible to use the *if* command inside these loops. To avoid this difficulty there is a way to translate these jumps to sequential codes. A simple example of this kind of translation would be the following; Let us suppose the following code fragment:

```
if (x>y)
        A=B;
else
        A=C;
end
```

It can be rewritten as:

```
Condition= x>y;
A=condition*B +~condition*C;
```

Other functions such as *break*, *return* or the ":"command inside a sum, must be replaced by their equivalent codes when found in *gfor* loops. Furthermore the *Jacket grand* command is not compatible with the generation of random numbers, so some parts of the code have assumed a structure change for optimal parallelization. A change in the denomination of the iterators of all *gfor* loops has been necessary. *Jacket* does not allow the direct use of the i and j iterators in these loops, since these are kept for complex numbers.

4.2 Experimental Results

4.2.1 Metrics

The experimental results presented in this section are based upon a series of tests executed in both CPU and GPU. These tests consist of a series of computing time measurements in both processing units. Times have been measured using *Matlab* software. Due to the stochastic nature of GAs, all experimental tests have been executed 30 times. Once obtained the required data, a graphic is presented to show and interpret the data in a simple way. The data used for graphs were obtained by the arithmetic average of all previous tests. A speed-up graphic is also included to evaluate the improvement of the execution time in the GPU.

Three Different CPU architectures (see table 4.1) has been used to compare the execution time with the GPU. These CPUs have been chosen due to their great variety of characteristics, for instance the P4 is the oldest CPU and has only capacity to execute one thread,

Table 4.1: *CPU Architectures used for comparison*

Processor	Intel Pentium 4	Intel Pentium SU4100	Intel Core i7-860
Number of cores	1	2	4
Number of threads	1	2	8
Max. Frequency	2.8 GHz	1.3 GHz	3.46 GHz
Cache	512 KB L2 Cache	2 MB L2 Cache	8 MB Intel Smart Cache
System Bus	533 MHz	800 MHz	2.5 GT/s
Operating system	Windows XP-32-bit	Windows 7 64-bit	Windows 7 64-bit
RAM -Memory	768 MB	4GB	8GB

Table 4.2: *Main characteristics of the GPUs used in the experiments*

Graphic Card	MSI nVidia 460 GTX OC	Gigabyte nVidia 570 GTX
CUDA Cores	336	480
Memory	768 MB	1280 MB
Clock for graphics	725 MHz	732 MHz
Clock for processor	1350 MHz	1464 MHz

whereas the i7-860 processor is a modern one with a capacity to execute up to 8 threads simultaneously. The SU4100 CPU is an intermediate architecture with a capacity to process two different threads simultaneously.

Jacket GPU programming is only compatible with *nVidia* graphic cards with CUDA technology. For the tests conducted here the GPU 460GTX and the 570GTX has been used. These are a modern hardware, a range normally used for entertainment and with prices of 300 and 150 euros respectively. These graphic card has been assembled in the third computer of the previous table, that is, the i7-860 CPU computer. Table 4.2 summarizes the main features of the GPU. The data previously presented on Figure 4.4 and Figure 4.6 have been obtained for the same architecture, corresponding to the third column in Table 4.1. Figures 4.7 and 4.12 have been executed only with the GPU 570GTX.

4.2.2 Execution Time analysis of the algorithm in the GPU

Figure 4.7 contains, once more, the 15 functions with the highest computation times, now when implementing the code on the GPU. As we can see observing this figure, the heaviest functions have changed. This is due to the *MEX* functions that *Jacket* makes using our code.

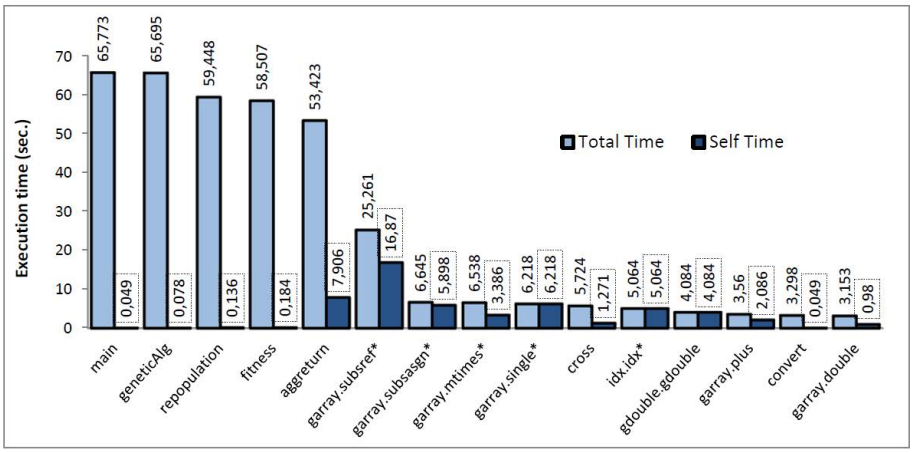


Figure 4.7: *Genetic algorithm execution in the GPU. It has been run for a company of 5000 individuals, 500 generation and the tournament selection algorithm Total Time= 66 seconds*

As mentioned above, those functions are written in C, so they become the best way to create an interface that converts the *Matlab* code *(. m)*, to CUDA, since these functions directly interact with the GPU. By implementing the code in the GPU the searched objectives have been achieved, the selection function no longer is a bottleneck in the program. On Figure 4.7, we cannot appreciate the time reduction in the selection algorithm, this is due to the fact that the *Jacket* parallelization has provoked the creation of new MEX functions that will be called from the selection algorithm. For instance *array subref* function is the one that consumes the greater part of the execution time. The program total execution time has been reduced, in this particular case, around 90% if compared to the CPU version with the same selection algorithm, and to a 97% if compared to the original algorithm.

4.2.3 Analysis of the execution time evolution

Figure 4.8 shows the related data with a set of executions where the parameter used to determine the number of individuals iteratively get more weight. These series of executions have been carried out with 500 generations. The number of individuals is represented on the x-axis. The increase intervals in this parameter are not identical along the graphic and raise

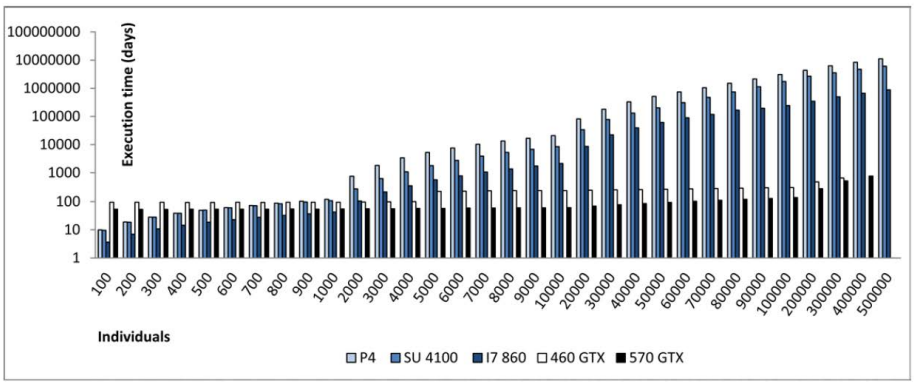


Figure 4.8: *Execution times (y- axis) for 500 generations and different number of individuals (x- axis)*

in an exponential scale each 9 executions. On the other hand the y-axis shows the execution time for the algorithm, and is represented in a 10-base logarithm scale. The compared elements are the different process units in which the AG has run, three CPU's and two GPU's. It is worthy to mention that for the CPUs, results obtained for 50000 individuals have been estimated accordingly to the average execution time of one generation.

Comparing the bars in Figure 4.8 we can calculate when the executed program in the GPU starts to get better results than in the evaluated CPUs. The program execution time is intrinsically linked to the number of individuals and the number of generations in a particular execution. The number of individuals is the main parameter which should focus our attention. It can be observed that for 100 individuals the execution of the algorithm in the GPU is slower, that is, the range of improvement that we get in the GPU's is not enough to exceed the speed at which the CPU executes the data. As the number of individuals increases, the improvement in the CPU compared to the GPU is reduced and finally is reached around at 600 (570GTX) and 900 (460GTX) for SU4100 CPU's and P4, and around 2000 individuals (both GPU's) with respect to i7-860. From these points, the implemented modifications start to provide beneficial in terms of computation time.

In the last execution series of Figure 4.8, in the GPU with around 400000 (460GTX) and

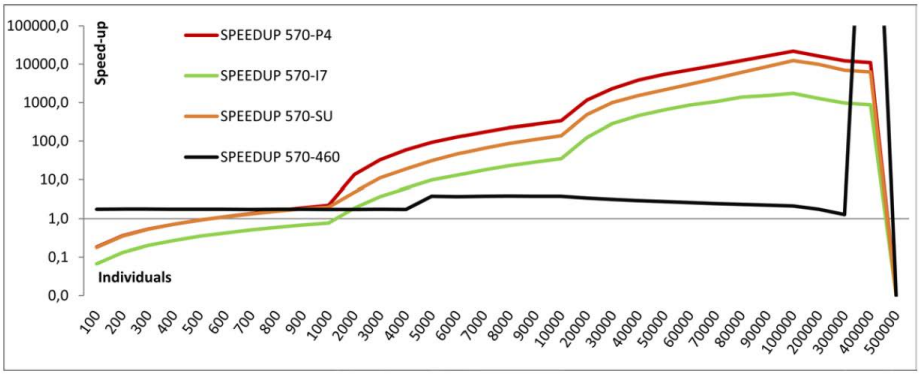


Figure 4.9: *Speed-up on the GPU (y-axis)with variable number of individuals (x-axis) and 500 generations*

500000 (570GTX) individuals, we observed that performance drops sharply (not showed for scaling). This sudden change is due to the fact that this amount of individuals the GPU memory capacity bound is reached, which is 768 MB for the 460GTX and 1280 MB for the 570GTX, however, these capacities of memory are not fully directionable for *CUDA*, so that the available memory is in fact lower. These amounts of memory are not very big, specially bearing in mind that most of the desk graphic cards have 1 GB of capacity and much more capacity if they are professional range.

Despite these limits, the population size of the GA reaches enough proportion for the execution of this particular problem. In fact with such a number of individuals to reach the convergence becomes an inconvenient. Actually, in our GAs, a large number of individuals cannot be a good choice in order to optimize this kind of problems. On next section, we will explain how to take advantage of a population with a high number of individuals without endangering the problem of convergence.

Figure 4.9 represents the speed-up obtained with the GA execution in the graphic card. It is based on the data obtained from Figure 4.8, so the parameters of individuals and generations are the same. In this figure the y-axis represents the speed-up for a certain number of generations. It can be observed from negative speed-ups (below 1) to very high improvements, around 10000 units. Highlighted in this chart the rise and sudden drop in

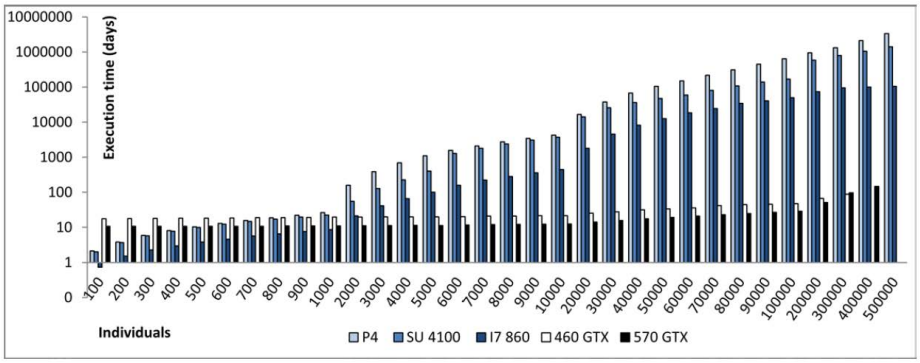


Figure 4.10: *Execution times (y-Axis) for 100 generations and different number of individuals (x-axis)*

the speedup of the 570GTX on 460GTX in the last two series of executions. The sudden rise is the lack of memory in the 460GTX, and the drop is the lack of memory of the 570GTX. The number of generations is the number of times that the algorithm will iterate. As no parallelization technique has been applied in this loop, its execution time will be in proportion to this parameter. This measure is not exact due to the fact that the size of the data also influences the program execution time. That is, the more GA generations, the more data will be stored by *Matlab*, since not only the final results are stored. At each iteration, *Matlab* stores the new array of fitness and the best individuals. Storing a large amount of data cause the slowdown of *Matlab* with the advance of the GA step.

In any case, regardless the number of generations of the GA run on the CPU, the improvement margin among the different CPUs will be the same. However, this situation does not occur in the GA execution over the GPU. For this implementation, and with this parallelization software and our strategy, the number of generations will influence the GPU speed-up on the different GPUs due to memory overloads in the GPU. Each additional iteration bears an extra cost, although sometimes small, what is true is that after a great number of generations it will have repercussions on the final execution time. To testify this event, another series of runs of the GA has been done with identical procedure that figures 4.8 and 4.9. Figure 4.10 and Figure 4.11 summarize the results of this set of experiments.

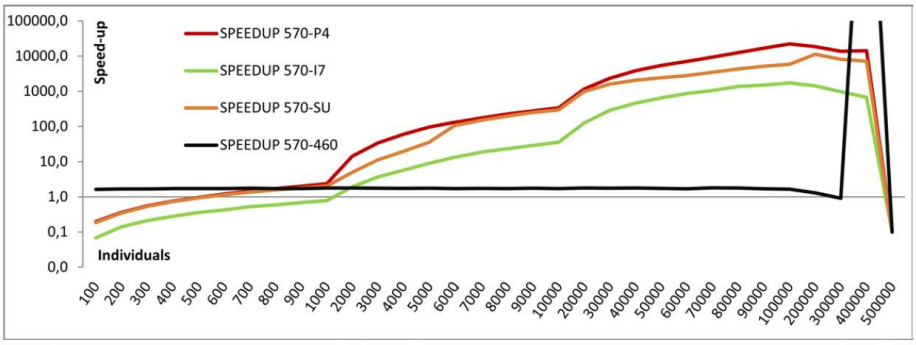


Figure 4.11: *Speed-up : Speed-up on the GPU (y-axis) with variable number of individuals(x-axis) and 100 generations*

Figure 4.10 (similar to the Figure 4.8) represents a series of executions on the different processing units. The parameters and representation of the axis are the same of Figure 4.8 . The only difference is the amount of executed GA cycles (100 in this case). As it can be observed comparing both figures, the execution times on CPUs are proportional. As an example, we can look at the value of the execution time for the $i7 - 860$ with 6000 individuals. For this number of individuals and 100 generations the value of the execution time is approximately 150 seconds (exactly 154.3 seconds). On the other hand, in figure the measure taken for the $i7 - 860$ with 6000 individuals is 775.92 seconds. If we multiply by five the execution time for 100 generations, we obtain approximately the same measure than in Figure 4.8 for 500 iterations. Nevertheless if we make the same operation for the GPU times, for example in the GPU 460 GTX, it would multiply 20.26 seconds 4.10 by 5, this way we should get result of about 100 seconds. However, in Figure 4.8 the execution time of the program is 219 seconds, with which it is proved the non-proportionality of the execution time when changing the number of generations.

Figure 4.11 shows the speed-up of time for the series of executions of Figure 4.10. As previously explained it can be observed that the non-proportionality of the generations impacts in the improvement margin of the GPU if compared to the CPUs, thus remaining a much better margin of improvement for 100 cycles (generations). So eventually we conclude

that for less number of generations and more than 2000 individuals the speed-up of the execution time on the GPU will be higher, and the higher the number of generations, the lower the improvement on time.

4.3 Taking advantage of the parallelization for trading the stock market

As it has been tested in the analysis of this GA, its implementation on the GPU allow us to keep a very high number of individuals without wasting execution time. However, the reader could have some concerns about the advantages of executing the GA with such a high number of individuals for our problem. If the number of generations has not increased in proportion to the number of the individuals we can lose the algorithm convergence and, even increasing the number of generations, could be that the algorithm converged with lesser number of individuals and generations, therefore missing valuable execution time, especially on real time operations.

To take advantage of generating a high number of individuals , we have changed the basic structure of the algorithm and a divided population has been implemented. As in the island model (10), this model maintains several independent populations, however, in this case we will never colaboration between populations, the populations are independent over the complete execution of the program. We can say it is a parallelization of the before parallelization. That is, for this problem the GA must be executed for about 250 companies per year, and with the data from the last twenty years, we need 5000 executions of the same GA with different data each time. To benefit from this characteristic, all the data from the companies in a given year are loaded and the population of the problem is divided by this number of companies. For example if we have 250 companies and we executed the GA with 125000 individuals (i.e a total population size of 125000 individuals) in which each company keeps 500 individuals.

It has been implemented in such way that all the basic processes of the GA as the selec-

tion or the crossover, is independent for each sub population. This way the consistency of the population is kept, allowing them to evolve independently. To avoid a population disproportionate to the number of companies the number of individuals fits automatically to the nearest multiple of the number of companies, so if there are 5000 individuals and 21 companies, the number of individuals the algorithm will implement will be the result of truncating (5000/21) and multiplying again by 21, total 4998 individuals.
Through this technique the number of executions can be simplified a lot, in the above mentioned example, the 5000 executions will be reduced to 20. Playing with the number of individuals and the number of companies that can be executed at the same time substantial advantage is achieved benefiting from the GPU capacity. Hence, that parallelization allows us to implement and test previous GA approximations. Regarding the investment results, the return obtained by the trading systems in the period 1986-2006 (on average for all the companies in the S&P 500 with available data) is clearly higher (870%) than the one reported by the same companies of the S&P 500 index in the same period (273%). Table 4.3 reports the results by year[1].

Figure 4.12, shows the 10 most expensive computationally functions of this particular executions. This test has been made simultaneously for the data in ten companies. To be able to compare with the previous execution time analysis, the same execution parameters have been used but rising the number of individuals up to 50000. In this way, each company is linked to 5000 individuals remaining the proper proportion for their direct comparison. The execution time for company is given by the total execution time divided by the number of companies, which makes a total of 10.9 seconds. The execution time for company is reduced approximately the sixth of the total, if compared to the parallelized version in the GPU presented on Section 4.2.

[1]These returns on historical data cannot be taken as a guaranty of future similar returns when using new data, since the rules implemented by the trading systems can suffer from over-fitting.

Table 4.3: *Returns obtained by the trading systems for the companies of the S&P 500 with available data during the period 1986-2006*

Year	# Companies	TD Returns	S&P return
1986	119	840%	311%
1987	138	1060%	350%
1988	136	746%	265%
1989	158	696%	251%
1990	158	684%	327%
1991	166	680%	209%
1992	176	793%	244%
1993	208	589%	188%
1994	214	613%	177%
1995	223	542%	155%
1996	217	518%	177%
1997	217	528%	204%
1998	215	589%	337%
1999	216	691%	386%
2000	215	953%	576%
2001	213	1043%	506%
2002	219	1265%	315%
2003	233	1281%	159%
2004	256	1500%	198%
2005	273	1461%	193%
2006	272	1209%	194%
Av. Annual Return		870%	273%

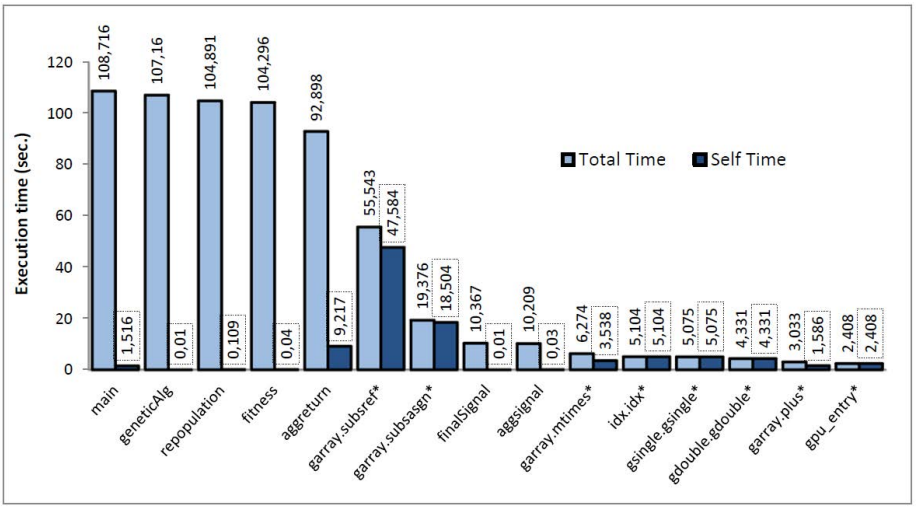


Figure 4.12: *Execution of the GA in the GPU. It is run for 10 companies, with 50000 individuals, 500 generations and tournament selection algorithm. Total time =109 seconds, partial time for company =10.9 seconds*

Chapter 5

Conclusions

In the last chapter of this document we will resume the most important point on this document, highlight de conclusions derivated of the work.

5.1 Grid computing versus GPU computing

We have used two different platforms for compute the same program and thus analyzed the benefits of the paralyzed architectures. Now, we going to compare the two parallelization technologies, *Jacket* and *BOINC*, the GPU and the grid.

First we will evaluate the difficult of develop one project in this platforms. On the one hand, developing one application for the *BOINC* platform requires a larger knowledge than the same develop for *Jacket*. Set up a application for *BOINC* request a middle level of C, SQL, Bash, etc. Nevertheless, you only need know the Matlab language to develop a application in *Jacket*, this means that the learning curve in *Jacket* is more soft than in the *BOINC* system.

On the other hand, if we already have an application not implemented in Matlab and we want profit the performance in the execution times, surely we should develop for *BOINC*. *BOINC* allows the parallelization in several languages, furthermore includes tools like wrapper for encapsulate applications in not compatibles languages.

We can affirm that the *BOINC* platform is far more stable and reliable. For example, the execution in this system can be stopped in any time and back to the execution ever we

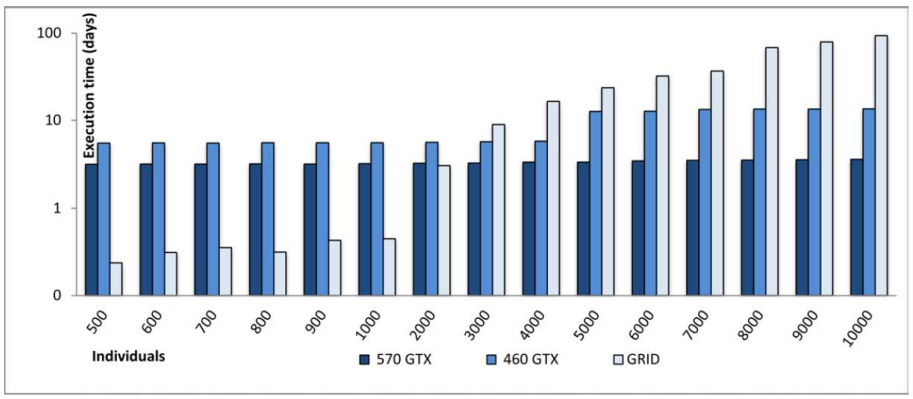


Figure 5.1: *Execution times comparative (y- axis) for 500 generations and different number of individuals (x- axis)*

want. The GPU cannot stop the execution. *BOINC* can send several works of the same company for comparing results, we can see the times of each execution or the percent of program executed. In short, *BOINC* is a mature platform that it have a countless options of configurations and facilities.

5.2 Performance

For talk about the performance, we compare bellow the execution times in several graphics.

Figure 5.1 shows a comparative graphic between the different execution times of the platforms analyzed in the present document. The y-axis represents the execution time measure in days. The x-axis shows the different amount of individuals in each execution. We use the fundamental analysis version of the program. However, the version in the grid system has a different selection algorithm. We use the wheel selection algorithm for the *BOINC* platform and the tournament selection for the GPU system.

Thus, we must take in consideration that the times in the GPU has a beneficial bonus for the change in this part of the code. The most important of this graphic is check the useful period of the executions; in other words, we can observe that the grid system becomes ineffi-

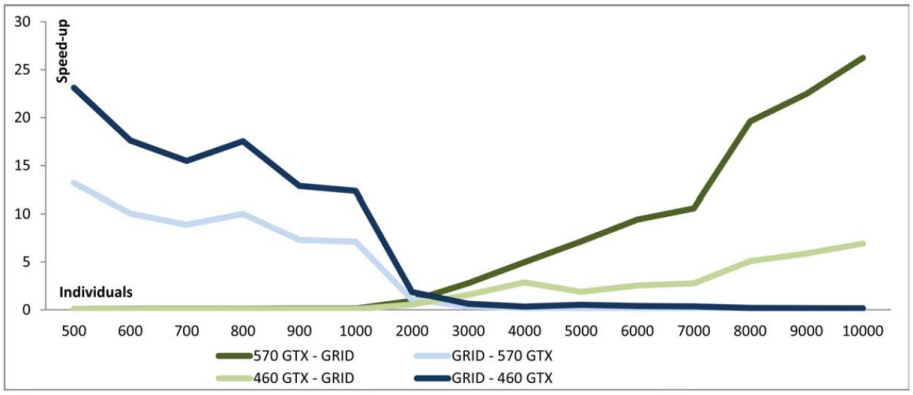


Figure 5.2: *Speed-up on the differents platforms (y-axis)with variable number of individuals (x-axis) and 500 generations*

cient around the 4000 individuals, because we consider that more than 10 days of execution starts to be too large times. Nevertheless, the GPU system remains with a little increments throughout the test (all the time bellow the 10 days).

With the same tests used in Figure 5.1,we have develop a graphic of speed-up. Figure 5.2 shows the speed-up between the GPU and the grid, and also the reverse form. More specifically, the dark lines represent the relations: 570GTX/grid and grid/570GTX. The clear lines represent the relations: 460GTX/grid and the grid/460GTX. As in the above figure, x-axis represents the amount of individuals and y-axis shows the speed-up.

The first thing that takes our attention is that both platforms could be useful in different periods. At the beginning, the *BOINC* system has a better behavior than the GPU, but this performance vanish when the number of individuals in a population rises. Beyond 2000 individuals the GPU/grid speed-up grows up constantly. Maybe the critical point (where both technologies have the same performance) should be shifted to the right because the GPU uses a different algorithm, however the trend to raise the speed-up by the GPU is clear.

After, we have analyzed the above figures; we can conclude that the grid system works fine

when the number of individuals is not too high.

5.3 Final conclusions

As we said above, the performance of investment decisions in stock markets is influenced by a huge number of variables related to macroeconomic, companies or market information that are difficult to analyze and even to follow since nowadays we have access to all this information (that is continuously changing) in real time. On the other hand, professionals making investment decisions suffer a high degree of stress due to the impressive amounts of money they manage. This factor may cause a bias in the analysis of the information by the trader.

The use of mechanical trading systems copes, at least partially, with these difficulties, since it avoids the psychological reactions of traders while allowing manage a huge amount of realtime data. The exponential growing complexity of the investment problem related with the number of factors affecting the investment performance makes it is necessary to count with powerful algorithmic tools to deal with the selection of indicators and parameters from the universe of existing economic and company variables and threshold values. GAs offer a powerful and fast search capacity due to its ability of processing information in a parallel way and the intelligent mechanism that is driving its functioning. Yet for dealing with intra day investment decisions or daily investment decisions for a big number of stocks is vital to speed up the GA process, to get in time good results. For this purpose, we carry out an innovative implementation of the GA that is fine-tuning the trading system, by means of using parallel computer architectures.

BOINC from *Berkeley* is a mature platform that achieve great results using a lot of computers. We can use old computers or the volunteer system for to cheapen the costs and we will obtain a powerful grid with few resources. The implementation of an application in *BOINC* requires a good knowledge of information technologies, this mean that it is a tool with high curve of learning.

Jacket of *Accelereyes* is a tool that brings results, recommended to decrease the difficulty of programming on GPU. By implementing the code in the GPU, the selection function no longer is a bottleneck and the total execution time has been reduced in a 65% if compared to the CPU version with the same selection algorithm, and to a 90% if compared to the original algorithm. There is a limit in the number of individuals implemented on the GPU of around 400000 individuals, where performance drops sharply.

Other important conclusion is that the number of generations will influence the GPU due to memory overload issues. Each additional generation suppose an extra cost, that after a great number of generations it will have repercussions on the final execution time.

We can also conclude that for less number of generations and more than 2000 individuals the speed-up of the execution in the GPU will be higher, and that for a higher number of generations the improvement margin will be reduce.

To take advantage of generating a high number of individuals a divided population has been implemented. For the targeted problem, the GA is executed for about 250 companies per year, and with the data from the last twenty years (5000 executions of the same GA with different data). To benefit from this characteristic, all the data of the companies in a given year are loaded and the population of the problem is divided by this number of companies. The execution time for company is reduced approximately to a 50% when compared to the parallelized version in the GPU.

We obtained 870% of profit for the S&P 500 companies (with available data) in the period 1986-2006, when using our trading systems to invest long and short-selling in the companies, that compares with a return given by these companies of 273% when remaining invested long during the same period.

5.4 Publications

1- Iván Contreras, Yiyi Jiang, J. Ignacio Hidalgo, Laura Núñez-Letamendia.

Using a GPU-CPU architecture to speed up a GA based real-time system for trading the Stock Market

Soft Computing - A Fusion of Foundations, Methodologies and Applications. Springer Berlin / Heidelberg. Issn: 1432-7643.

doi: 10.1007/s00500-011-0714-3.

2- Iván Contreras, Yiyi Jiang, Laura Núñez-Letamendia, J. Ignacio Hidalgo.

Arquitectura GPU-CPU para la aceleración de un AG en un sistema de inversión bursátil en tiempo real. Congress on numerical methods in engineering.

Coimbra - Portugal. 14-17 June, 2011

http://www.itecons.uc.pt/cmne2011/uk/introducao.htm

Bibliography

[1] Accelereyes. *Jacket Documentation*, February 2011.

[2] F. Allen and R. Karjalainen. Using genetic algorithms to find technical trading rules. *Journal of Financial Economics*, 51(2):245 – 271, 1999.

[3] T.G. Bali, O. Demirtas, and H. Tehranian. Aggregate earnings, firm-level earnings, and expected stock returns. *JFQA*, 43(3):657–684, 2008.

[4] W. Banzhaf, S. Harding, W. B. Langdon, and G. Wilson. Accelerating genetic programming through graphics processing units. In *Genetic Programming Theory and Practice VI*, pages 1–19. Springer, 2009.

[5] S. Basu. The investment performance of common stocks in relation to their price-earnings ratios: A test of the efficient market hypothesis. *Journal of Finance*, 32:663–682., 1977.

[6] Berkeley. *Url Berkeley*, April 2011.

[7] Berkeley. *Url Boinc options*, 2011.

[8] Berkely.

[9] Campbell and Yogo. Efficient tests of stock return predictability. *Journal of Financial Economics*, 81:27–60, 2006.

[10] Erick Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[11] L.K.C. Chan, Y. Hamao, and R. Lakonishok. Journal of finance. *Fundamentals and stock returns in Japan*, December:1739–1764, 1991.

[12] Dyndns.

[13] EMCO.

[14] E. Fama and French. The cross-section of expected stock returns. *Journal of Finance*, 47(2):427–465., 1992.

[15] E. F. Fama and K. R. French. Business conditions and expected returns on stocks and bonds. *Journal of Financial Economics*, 25:23–49, 1989.

[16] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Boston, MA, USA, 1st edition, 1989.

[17] http://setiathome.berkeley.edu/.

[18] Intel.

[19] Y. Jiang and L. Núñez. Efficient market hypothesis or adaptive market hypothesis? a test with the combination of technical and fundamental analysis. In *Proceedings of the 15th International Conference. Computing in Economics and Finance.* University of Technology, Sydney, Australia., The Society for Computational Economics, July 2009.

[20] F. Krüger, O. Maitre, S. Jiménez, A. Baumes, and P. Collet. Speedups between 70 and 120 for a generic local search (memetic) algorithm on a single gpgpu chip. In *EvoApplications (1)*, pages 501–511, 2010.

[21] W. B. Langdon. A fast high quality pseudo random number generator for nvidia cuda. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO'09, pages 2511–2514, New York, USA, 2009. ACM.

[22] O. Maitre, L. Baumes, N. Lachiche, A. Corma, and P. Collet. Coarse grain parallelization of evolutionary algorithms on gpgpu cards with easea. In *Proceedings of the*

11th Annual conference on Genetic and evolutionary computation, GECCO '09, pages 1403–1410, New York, NY, USA, 2009. ACM.

[23] Mathworks. *Manual of Matlab*, 2011.

[24] Brad L. Miller and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.

[25] A. Munawar, M. Wahib, M. Munetomo, and K. Akama. Hybrid of genetic algorithm and local search to solve max-sat problem using nvidia cuda framework. *Genetic Programming and Evolvable Machines*, 10:391–415, December 2009.

[26] L. Núñez. Fitting the control parameters of a genetic algorithm: an application to technical trading systems design. *European Journal of Operational Research*, 179:847–868, 2007.

[27] Laura Núñez. Trading systems designed by genetic algorithms. *Managerial Finance*, 28:87–106, 2002.

[28] Laura Núñez, J. Pacheco, and S. Casado. Applying genetic algorithms to wall street. *Int. J. Data Mining, Modelling and Management*, forthcoming:in press, 2011.

[29] nVidia. *Compute Unified Device Architecture Reference Manual*, 2011.

[30] Petr Pospichal, Jirí Jaros, and Josef Schwarz. Parallel genetic algorithm on the cuda architecture. In *EvoApplications (1)*, pages 442–451, 2010.

[31] M. Reinganum. Selecting superior securities charlottesville. the tesearch foundation of the institute of chartered financial analysts. Technical report, The Tesearch foundation of the institute of Chartered Financial Analysts., 1988.

[32] Jay R. Ritter. Behavioral finance. *Pacific-Basin Finance Journal*, 11(4):429–437, 2003.

[33] John R.Koza. *Genetic Programing*. Bradford Books, 1996.

[34] Grid Systems. *Grid Systems*, 2011.

[35] VirtuaBox. *VitualBox Documentation*, April 2011.

[36] VMware. *Url Mware*, April 2011.

[37] Sifa Zhang and Zhenming He. Implementation of parallel genetic algorithm based on cuda. In *Proceedings of the 4th International Symposium on Advances in Computation and Intelligence*, ISICA '09, pages 24–30, Berlin, Heidelberg, 2009. Springer-Verlag.

Appendix A

Install a Boinc server step by step

A.1 Introduction

The first step is to have installed a stabled version of Linux. The Debian distribution is the version most used for this propose. Therefore we will install a Boinc server more easy if we use one version of Debian.

The Boinc server as compiled from the Boinc sources does not perform directly. That Boinc server is "only" the skeleton for the real server.

In the following, please distinguish
1.the folder to which the files of the BOINC server templates install ('/usr/share/boinc-server') 2.the name of the Debian package providing those files ('boinc-server-maker') 3.the location of your very own project ('you/name/it')

The preparation from the Debian sources is not too different from a preparation from the original source tree. Debian should just shorten the process to a first success.

A.2 Install the Boinc sources

Install BOINC Server Template files

Install BOINC server dependencies :

```
sudo apt-get install subversion build-essential
apache2 php5 mysql-server php5-gd php5-cli php5
-mysql python-mysqldb libtool automake autoconf
 pkg-config libmysql++-dev libssl-dev
```

Then, we create a new user and a new group for Boinc, with them managed Boinc. Furthermore we add www–data to that group :

```
sudo addgroup --system boincadm
sudo adduser www-data boincadm
```

We download the Boinc source from internet :

```
svn co http://boinc.berkeley.edu/svn/branches/server_stable boinc
```

Later, we need to compile the software Boinc :

```
cd boinc
./_autosetup
./configure --disable-client
make
```

Installation with the Debian server package

If we have used a Debian versión we only need to install the boinc-server-maker package from unstable :

```
sudo apt-get install boinc-server-maker
```

A.3 Project-specific configuration

The preparation of the final project from the Boinc server template is a manual process. That should be applied only once for every project and then for updates of the Boinc server sources. On the other hand, the Boinc server as compiled from the Boinc sources does not

perform directly.

The following needs to be performed for every project, with or without the Debian Boinc server package. In the following we will work on a single Bash shell. This allows us to use mostly self-explanatory variables to help your local adaptation.
To help working consistently on multiple shells, we need a place to redirect the parameters of the project configuration. We do that only to help ourselves; the Boinc code will not use it.
The first part of this initialisation of the configuration start of a project create the file that the project parameters shall be stored in.

```
echo > \~/.boincproject.sh
chmod 600 \~/.boincproject.sh
```

After, we need configure the parameters for the MySQL database:

```
cat << EODBCONFIG >> ~/.boincproject.sh
# password for write access
pw=ThePaswordForMySQLDataBase
# name of the MySQL database
dbprojectname=boinctest
EODBCONFIG
```

With the next code we will create a MySQL database for the Boinc projects, with the variables defined before :

```
# read config if available
[ -r ~/.boincproject.sh ] && . ~/.boincproject.sh
if ! echo "DROP USER 'boincadm'@'localhost'"
     | mysql -u root -p; then
            echo "If the removal of the previous
              user fails because the user is not
              existing, then this does not matter.
              Other errors would be required a manual
              removal."
fi

if   [ -z "\$dbprojectname" ]; then
            echo "Variable 'dbprojectname' not set";
elif [ -z "\$pw" ]; then
            echo "Variable 'pw' not set";
else
            # piping commands to mysql shell
```

```
            cat <<EOMYSQL | mysql -u root -p mysql
            DROP DATABASE IF EXISTS
                    \$dbprojectname;
            CREATE DATABASE IF NOT EXISTS
                     \$dbprojectname;
            CREATE USER 'boincadm'@'localhost'
                     IDENTIFIED BY '\$pw';
            GRANT ALL PRIVILEGES ON \$dbprojectname.*
                     TO 'boincadm'@'localhost';
            EOMYSQL
fi
```

When we are arrived here, we should be able set up the first project. In the next chapter we explain how create and execute a projet.

Appendix B

Boinc project step by step

After the server installation, we can create a project. As in the Boinc server installation, depending on if we use the Debian package or source Boinc tree, we will have two ways for create a project (only in some parts).

B.1 Create a Boinc project

We will insert the next code in a shell for create a project. We need a static IP, If you only have dynamic IP addresses you can use services like 12.

```
cat <<EOCONF >> ~/.boincproject.sh
# Your server's IP adress
hosturl=http://a.b.c.d
# Full project name
fileprojectname=\$dbprojectname
# Friendly project name
niceprojectname="BoincTestProject@Home"
# Location at which sources shall be kept
installroot=/var/tmp/boinc
EOCONF
```

Maybe we need edit the script boincproject.sh with a text editor, because some linux distributions may however have that directory cleaned at boot time. The installation is different for the debian package and the tree source. Later, in the section B.1 the steps will are the same again.

Installation from within the Boinc source tree

```
[ -r ~/.boincproject.sh ] && . ~/.boincproject.sh
if [ -z "\$installroot" -o -z "\$hosturl" -o -z
                    "\$dbprojectname" -o -z "\$pw" \-o -z
                    "\$niceprojectname" -o -z
                    "\$fileprojectname" ] ;then
    echo "Missing configuration parameter."
else
    [ -d "\$installroot" ] || sudo mkdir -p "\$installroot"
    sudo ./tools/make_project --url_base "\$hosturl"
    --db_name "\$dbprojectname" \ --db_user boincadm
    --db_passwd "\$pw" --drop_db_first \ --project_root
     /var/www/boinc/\$fileprojectname \ "\$fileprojectname"
      "\$niceprojectname"
fi
```

Installation with the Debian server package

If we use boinc-server package, we will need to use this command instead:

```
[ -r ~/.boincproject.sh ] && . ~/.boincproject.sh
[ -d "\$installroot" ] || sudo mkdir "\$installroot"
sudo PYTHONPATH=\$PYTHONPATH:/usr/share/pyshared/Boinc/
/usr/share/boinc-server/tools/make_project --url_base
 "\$hosturl" --db_name "\$dbprojectname" --db_user
 boincadm --db_passwd "\$pw" --drop_db_first
 --project_root "\$installroot"/"\$fileprojectname"
 --srcdir /usr/share/boinc-server/ "\$fileprojectname"
 "\$niceprojectname"
```

When these commands are inserted, we should answer yes to all questions.

Adjusting permissions for project directory

Now, we need to change files and directories permission with the next command:

```
if [ -z "\$installroot" -o -z "\$fileprojectname" ]; then
  echo "Not all variables are set for the configuration"
  echo "Error, do not continue."
elif [ ! -d "\$installroot"/"\$fileprojectname" ]; then
  echo "The directory '\$installroot/'\$fileprojectname'
                is not existing"
  echo "Error, do not continue."
else
  cd "\$installroot"/"\$fileprojectname"
  sudo chown root:boincadm -R .
  sudo chmod g+w -R .
  sudo chmod 02770 -R upload html/cache
                                        html/inc html/languages \
```

```
            html/languages/compiled
            html/user_profile
fi
```

If the system asked for the user sudo password, the execution was successful.
If those permissions are sufficient depends on the user the the web service runs with. For Debian, running apache as user www-data, the include files need to be accessible. Those they should be with www-data added to the group boincadm with those files group accessible and group writable. However, there seem to be some remaining issues. To circumvent them, we run the next commands:

```
if [ -d html/inc -a -d cgi-bin ]; then
  sudo chmod    o+x html/inc
  sudo chmod -R o+r html/inc
  sudo chmod    o+x html/languages/
  sudo chmod    o+x html/languages/compiled
else
  echo "You are not in your project directory"
fi
```

The "-r" is important to reach all the files since the directory cannot be read before the execution of "sudo". The next step is prepare a automated restart upon failure. We need to add to the project cronjob:

```
sudo crontab -e
```

Then, we add this line to editor appear after run command above:

```
0-60/5 * * * * /var/boinc/testproj/bin/start --cron
```

We verify that the cron entry was indeed accepted:

```
sudo crontab -l
```

Config password for Boinc server administrative page:

```
sudo htpasswd -c html/ops/.htpasswd USERNAME
```

We need to config Apache to call Boinc server:

```
sudo cp \${fileprojectname}.httpd.conf
    /etc/apache2/sites-available/
sudo a2ensite  \${fileprojectname}.httpd.conf
sudo /etc/init.d/apache2 reload
```

Web interface

Each project has a specifical web interface, is we have follow the steps, the web site should now be accessible. It should be available as an alias "$fileprojectname". If we look in "/etc/apache2/sites-enabled/$filesprojectname.httpd.conf", we will find the automated home page to need some further manual adjustments.
If the home page does not show up then there may be some missing file or some bogus file permission. We can read the file located in "/var/log/apache/error.log" should give us sufficient clues to fix the issue.

With a typical Debian apache setting, the expected Boinc web interface only shows up after clicking on index.php.
The web pages show whenever there should be the project name the string "REPLACE WITH PROJECT NAME". To fix this and other things like your public email address, edit html/project/project.inc .

Configure and start Boinc daemons

When we arrive to this point, we will need to configure and start the Boinc daemons. After of this, we have a redady Boinc server for the execution of our projects.

The tool xadd configures the project by parsing the 'project.xml' file. That 'project.xml' file initially only lists the most common platforms. Then it mentions the application that should be supported, which is 'uppercase'.

```
sudo bin/xadd
```

The $(arch) identifies the platform that this installation is performed on and consequently that the binary in */usr/lib/boinc-server* is compiled for. The information on the availability of that binary needs to be communicated to the database by executing the *update_versions* tool by a mere.

```
sudo PYTHONPATH=/usr/share/boinc-server/py
        bin/update\_versions.
```

Now, we can already start the daemons:

```
sudo bin/start
```

Running start, this should show something like:

```
Entering ENABLED mode
Starting daemons
  Starting daemon: feeder --d 3
  Starting daemon: transitioner --d 3
  Starting daemon: file\_deleter --d 3
```

Appendix C

Create a executable for Matlab

We can use the *Matlab* executables for to do one application independent of *Matlab*. Thus, we do not need a *Matlab* installation for running these executables, however we need the MCR library. MCR (*Matlab* Component Runtime) is a free set of libraries that allow for running our applications without *Matlab* . We follow the next steps for create the necessary files:

First, we must open Matlab and we execute the next command:

```
>>deploytool
```

Matlab opens a new panel with differents options, this options depends a little on the version used and on the platform where Matlab is installed.

We must select the option that allows us create a new project. The application requests the name of the project, which will be the name of the executable. Then we select the option the "Standalone Application".

Now, we look for the option "add" -> "main function", so we select our main fuction, that it must have the next structure:

```
Function main ("input parametres") "body function"
```

Furthermore, with the option "add" -> "Other files" we add all necesary files for the properly works of the application (for example .m or .mex).

After we do the above steps, we are ready to create the executable with the option "Build the project". For this step, *Matlab* needs a C compiler. Thus, maybe we need set up one

compiler. By defect, a configurated C compiler is provided by *Matlab*, his name is Lcc. If *Matlab* request a C compiler we insert the next command for select it: One time that we do the lasttest steps, we will find the executable in the address:

```
Current Directory/ProjectName/Distrib\\
```

We recall that in the case of Linux, in addition of the executable, the software will create a sh script, that will be necessary for the execution of our application. This script has the functionality to enable the MCR library environment and run our application.